26862

National Library of Canada | Bibliothèque nationale du Canada

CANADIAN THESES ON MICROFICHE | THÈSES CANADIENNES SUR MICROFICHE

NAME OF AUTHOR/*NOM DE L'AUTEUR* KENNETH HENRY NEWMAN

TITLE OF THESIS/*TITRE DE LA THÈSE* THE EFFECTS OF MEMORY ARCHITECTURE IN MULTIPROCESSOR DESIGN

UNIVERSITY/*UNIVERSITÉ* UNIVERSITY OF ALBERTA

DEGREE FOR WHICH THESIS WAS PRESENTED/ *GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* M. Sc.

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE GRADE* 1975

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* DR. JOHN TARTAR

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/*DATÉ* 21 OCTOBER 1975 SIGNED/*SIGNÉ* Ken H. Newman

PERMANENT ADDRESS/*RÉSIDENCE FIXÉ* 116 WESTOVER DR. S.W.
CALGARY, ALBERTA

NL-91 (3-74)

THE UNIVERSITY OF ALBERTA

EFFECTS OF MEMORY ARCHITECTURE IN MULTIPROCESSOR DESIGN

by

Kenneth Henry Newman (C)

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1975

T H E U N I V E R S I T Y O F A L B E R T A

RELEASE FORM

NAME OF AUTHOR: Kenneth Henry Newman

TITLE OF THESIS: Effects of Memory Architecture in Multiprocessor Design

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1975

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed) Ken H. Newman .....

PERMANENT ADDRESS:
116 Westover Dr. S.W.
Calgary, Alberta
Canada

DATED 13 October 1975

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled "Effects of Memory Architecture in Multiprocessor Design", submitted by Kenneth Henry Newman in partial fulfilment of the requirements for the degree of Master of Science.

John Tartar
Supervisor

W. S. H. Davis

W. K. Dawson

J. Hogarth

Date October 6, 1975

ABSTRACT

Computer architecture study is prompted by a number of requirements, one of which is greater processing "power". One measure of "power" is the instruction execution rate (IER). Although higher execution rates have been achieved through increased logic speeds, processing demands still outstrip these gains necessitating the study of organizations which provide higher IER. The central themes of this thesis are the development of methods for evaluation of such organizations, and the application of the methods to the study of three multiprocessor architectures.

Computer architecture consists of both hardware and software. In this study, a software design is proposed which permits the separation of instructions and data, allowing each to be stored in different (disjoint) memory modules. The effects of both disjoint and non-disjoint memory architecture upon the IER and resource utilization in multiprocessors are studied by developing analytic and simulation models. The analytic models relate the IER to memory and processor speeds, their number and their interconnection. Simulation models are developed to provide more insight into the architectures. The importance of both types of models as design tools is illustrated by specific examples. Three architectures are compared with respect to IER and utilization.

ACKNOWLEDGMENTS

I would like to thank Dr. J. Tartar, my supervisor, for his support and guidance throughout the preparation of this thesis. I thank my wife, Kathy, for her moral and technical support and for her determination to see this thesis completed. A special note of thanks to Mom and Dad Fyfe; without their help, the preparation of this thesis would have been very difficult. Lloyd Benbow is gratefully acknowledged for his technical assistance that allowed the timely completion of the thesis.

TABLE OF CONTENTS

I. INTRODUCTION 1

1.1 Scope of Study 1

1.2 Objectives 2

1.3 Terminology 4

1.4 Notation 5

II. BACKGROUND 7

2.1 Classification of Computer Organizations 7

2.2 SISD Machine Organization 10

2.3 MIMD Machine Organization 16

2.3.1 Overview 16

2.3.2 Multiple Processor Computers 18

2.4 MIMD Parameters 22

2.4.1 Processor Parameters 22

2.4.2 Memory Parameters 24

2.4.3 Processor-Memory Switching Parameters ...24

2.5 Previous Evaluation Attempts 25

2.5.1 Analytic Techniques 25

2.5.2 Simulation Techniques 26

2.5.3 Other Approaches 27

2.6 Summary 28

III. ANALYTIC MODELS OF CERTAIN MIMD DESIGNS 29

3.1 Introduction 29

3.2 Instruction Timing Diagram 32

3.3 SISD Computer with Buffering 34

3.4 SISD Computers with Disjoint Memories 41

3.5 MIMD with Dedicated Program Memories 43

3.6 Generalized MIMD with Disjoint Memories 47

3.7 Summary 54

IV. SIMULATION AND RESULTS 55

4.1 Introduction 55

4.2 The Simulation Model 56

4.3 Instruction Execution Rate Results 60

4.3.1 Analytic and Simulation Results 60

4.3.2 Results using Single Address Format 65

4.3.3 Instruction Mix Results 78

4.4 Utilization Results 95

4.5 Summary 105

V. CONCLUSION 108

5.1 Conclusions 108

5.2 Suggestions for Further Research 111

REFERENCES 114

LIST OF FIGURES

3.1 MIMD with Disjoint Program and Data Memories 31

3.2 Instruction Timing Diagram 33

3.3 SISD with Instruction Buffering and Prefetch 34

3.4 ITD for SISD Computer with Overlap 35

3.5 Execution Times for ITD of Figure 3.4 35

3.6 ITD for SISD Computer with Disjoint Memories 42

3.7 MIMD Design with Dedicated Program Memories 43

3.8 ITD for MIMD with Dedicated Program Memories 44

3.9 Simplified ITD for Figure 3.8 44

3.10 IER for MIMD with Dedicated Program Memories 46

3.11 Instruction Execution Flow with Disjoint Memory .48

3.12 ITD for MIMD Computer with Disjoint Memories 48

4.1 Simulator Flow Diagram 57

4.2a MIMD Performance with Disjoint Memories 61

4.2b MIMD Performance with Dedicated Memories 64

4.3a Disjoint vs Mixed Memory Performance (tp=0) 66

4.3b Disjoint vs Mixed Memory Performance (tp=tw) 67

4.4 MIMD Performance with Disjoint Memories (varying processor speeds) 70

4.5a Disjoint vs Dedicated Memory Performance(tp=0) ..72

4.5b Disjoint vs Dedicated Memory Performance(tp=2tw) 73

4.6 MIMD Performance with Dedicated Program Memories (varying processor speed) 76

4.7 Instruction Classes 79

4.8a Results for Instruction Mix 5 85

4.8b Results for Instruction Mix 6 86

4.8c Results for Instruction Mix 7 87
4.9a Mixed Memory Performance for Different Mixes 90
4.9b Disjoint Memory Performance for Different Mixes .91
4.9c Dedicated Memory Performance for Different Mixes 92
4.10a IER and Utilization with Mixed Memory 101
4.10b IER and Utilization with Disjoint Memory 102
4.10c IER and Utilization with Dedicated Memory 103

LIST OF TABLES

4.1 Instruction Mixes 80

4.2 Relative IER for Mixes 1 to 4 81

4.3 IER and Utilization in Balanced Configurations ...97

4.4 IER and Utilization Ranges for Different Mixes ...98

LIST OF SYMBOLS

D	Data Operator
E'(t)	mean instruction execution time (without branches)
E(t)	mean instruction execution time (with branches)
Kc	Control Unit
k	number of program memories
m	number of data memories
M	mixed memory
Md	data memory
Mp	program memory
Mib	instruction buffer memory
n	number of processors
P,Pc	processor
Pb	probability of branch instruction occurring
Pd	probability that processor is queued on Md
Pp	probability that processor is queued on Mp
RD	rate of operand fetches
RP	rate of instruction fetches
ta,tad,tap	access times: mixed, data and program memories
tc,tcd,tcp	cycle times: mixed, data and program memories
td	instruction decode time
tei	processor execution time for instruction i
ti	elapsed execution time for instruction i
tp	mean processor execution time
tw,twd,twp	restore time: mixed, data and program memories
tr	instruction buffer access time
ts	switch delay

CHAPTER 1

INTRODUCTION

1.1 Scope of Study

Although the potential of parallel processing and multiprocessing has been recognized since the beginning of electronic computers, interest in theses machines has risen sharply in the last decade. The study of multiprocessing is often divided into hardware and software research although they are complementary. In this thesis, both hardware and software architecture are considered as certain multiprocessor designs are proposed and studied. These designs consist of specific hardware configurations and a software arrangement which supports the hardware.

Hardware architecture is considered to consist of technology and organizational concepts. While technology plays an important role in computer architecture, it is relegated a secondary position to organization in this thesis. The organizational concepts involved in computer design are studied at a high level--the Processor-Memory-Switch (PMS) level--and the inter-relations of these major components are analyzed as they effect processing power through parallelism. While software research has included recognition of parallelism within existing programs, extension or design of languages for expressing parallelism and algorithm design for

exploiting implicit parallelism within problems, this thesis assumes the existence of parallel processable instruction streams and proposes a simple software organization which supports the hardware architectures proposed.

1.2 Objectives

The major objectives are:

(1) development of analytic models to describe instruction execution rates for certain multiprocessor designs,

(2) evaluation of these analytic models through comparison to simulation studies, and,

(3) further study of the execution rates and resource utilization of the designs for various configurations and instruction mixes, and,

(4) to illustrate how the results obtained for the above objectives may be utilized in design procedure to evaluate multiprocessor configurations that are optimum with respect to certain design criteria.

A review of some parallel and multiprocessor projects is contained in Chapter 2 in order to establish a background for multiprocessor study. Uniprocessors are briefly discussed as it is the limitation of these machines which helps to prompt multiprocessor study. To study these machines, they are first classified according to generic type. Four types are recognized based on the 'stream' concept: single-instruction

stream, single-data stream (SISD); single-instruction stream, multiple-data stream (SIMD); multiple-instruction stream, single-data stream (MISD); and, multiple-instruction stream, multiple-data stream (MIMD). Each type of machine is reviewed to determine the major parameters which may be used to describe its performance (execution rate).

Once multiprocessing systems are classified according to the scheme above and reviewed with respect to the criteria mentioned, Chapter 3 develops analytic models which describe the instruction execution rates (IER) for certain multiprocessor (MIMD) designs under the assumption that the multiprocessor is executing one type of instruction -- a single address type. Two reasons for developing analytic models are to gain quantitative understanding of factors which govern multiprocessor performance (analysis) and to aid in the design of multiprocessor system (synthesis).

In Chapter 4, the analytic models are tested by comparison to simulation results for the designs. Further simulation results using various instruction mixes are obtained to test the effects of the assumptions made in Chapter 3. Examples of how these models may be used in the design of multiprocessor systems are given. Chapter 4 also reports results of resource utilization studies. The last chapter, Chapter 5, summarizes the results obtained and points out areas where further research is needed.

1.3 Terminology

A multiprocessing system is considered to consist of two or more generalized processing units, of the "central processing unit" type, which obtain their instructions and data from one or more memory modules (memories) via either a switched or non-switched link. A memory which is connected to a processor (Pc) through a non-switched link is termed dedicated (to that Pc) while a memory that may be connected to more than one Pc (not at the same time) is said to be a switched memory. Whenever several processors, nPc, are connected by a switch to several memories, mM, it will be assumed that the switch allows a minimum number of simultaneous conversations equal to min(m,n). That is, the concurrency of the switch is min(m,n).

In this thesis, software architecture is included in the study of hardware architecture to the extent that hardware architectures are proposed which require the separation of instructions and their data into separate segments which may then be physically stored in different memory modules. Such a software arrangement is not uncommon and is often referred to as producing "pure procedures" [30]. Pure procedures appear to have originated in order to allow re-entrant and shareable program segments but for the purpose of this thesis, this arrangement allows instruction fetches to be made from one memory module and operand fetches from another memory module. A set of memories which supports such software design are

termed disjoint in this study. A memory system in which instructions and data are intermixed in any module (non-disjoint) is termed a mixed memory.

1.4 Notation

The descriptive notation used throughout this thesis is the PMS system developed by Bell and Newell [8]. In PMS, there are seven basic component types, each distinguished by the operations it performs. The major components used here are:

Memory,M. A component that holds or stores information over time without being changed in any way. The notation, Mp, is used to refer to a memory which contains only programs (instructions), while Md refers to a memory containing only operands or data. The letter, M, without qualifier refers to a mixed memory system.

Switch,S. A component that constructs a link between other components. Each switch has associated with it a set of possible links, and its operations consist of setting some of these links and breaking others. For a multiprocessor system, the concurrency of the switch must be greater than one.

Processor,P. A component that is capable of interpreting a program in order to execute a sequence of operations. It

consists of a set of operations of the types above (plus others not defined here) plus the control necessary to obtain instructions from a memory and interpret them as operations to be carried out. Various types of processors are possible: Pc (central processor), Pio (input-output processor), P.display (graphics display processor), etc.

Other notations such as 'K' for control unit are defined where used in the text.

CHAPTER 2

BACKGROUND

The main purpose of this chapter is to describe multiple processor designs and identify the component parameters which best describe their "performance". These parameters are used in developing analytic models in Chapter 3 and as simulation parameters in Chapter 4. A review of previous evaluation attempts for multiprocessors is also included.

2.1 Classification of Computer Organizations

Two problems that can be recognized when trying to classify computer organizations are the choice of the level of description and the isolation of the description from the problems the organization is designed to solve. One approach to these problems can be seen in [8] where various levels of description are identified and a classification scheme proposed. A second taxonomy based partly on the computer organization and partly on the computer function is proposed by Baer [2,3]. In this thesis, a fairly high level distinction between machines is made based on Flynn's "stream" concept [24]. A stream is a sequence of either instructions or data, ie., items that are either executed or operated upon in the conventional sense. Computer organizations can be "categorized by the magnitude, either in space or time

multiplex, of interactions of their instruction and data streams" [26]. Only two levels of magnitude -- single stream and multiple stream -- are essential to adequately classify the types of machines discussed.

With two levels of magnitude and two types of streams, four broad classifications can be made:

(1) SISD: single-instruction stream, single-data stream organization. Most conventional machines fall into this class. The characteristics of these machines are developed in Section 2.2. This class is included to show the development of computer systems and serves as a basis with which to compare multiprocessing computer organizations.

(2) SIMD: single-instruction stream, multiple-data stream organization. Machines in which one instruction operates on many operands simultaneously fall into this class. Three subclasses of organization are identified: array processors such as SOLOMON [13] and ILLIAC IV [5,14,48]; pipeline processors like the Control Data STAR-100 [36,40] and the Texas Instrument ASC [70]; and associative processors like the Goodyear STARAN [1]. While these designs offer increased execution rates by implementing special types of parallelism, they are best suited for processing particular classes of problems. Because of this lack of general applicability, these designs are not discussed further.

(4) MISD: multiple-instruction stream, single-data stream

organization. This type of machine as described by Flynn [26] encompasses "specialized streaming organizations using multiple-instruction streams on a single sequence of data and the derivatives thereof. The plugboard machines of a bygone era were a degenerate form of MISD wherein the instruction streams were single instructions, and a derived datum (SD) passed from program step i to program step i+1 (MI)". This organization might be useful for primitive functions, eg. Boolean operations, on data or some specialized combinations of higher level functions, eg. polynomial evaluation, but seems very restricted for generalized applications. Because this organization is not common and is quite limited in application, it is not considered further in this thesis. It is mentioned here only for completeness.

(3) MIMD: multiple-instruction stream, multiple-data stream organization. This class includes those machines variously referred to as multiple processor computers, multi-processors, true multi-processors, shared resource multi-processors, network computers, distributed computer systems, distributed control networks, tessellated computers, computer networks and possibly a few more. In Section 2.3, an effort is made to reduce the number of sub-classes and discuss the more important designs in detail.

2.2 SISD Machine Organization

The SISD organization is the forerunner of more advanced parallel structures. It is this organization which is most common today, is perhaps the easiest to understand, and can be used as a basis with which to compare other organizations. SISD organizations have been the subject of much research with the result that two types of machine can be recognized: the classical von Neumann machine and SISD machines which utilize overlap mechanisms.

The history of computer development in the U.S.A. is described by Rosen [64] and by Bell and Newell [8]. The first electronic computers seem to have evolved directly from mechanical and electro-mechanical calculators in an effort to increase calculation speeds. The problems to be solved by these machines were mainly those involving sequential calculation of simple scientific algorithms.

Scientific applications have two major requirements [8]. The first is great numerical accuracy which is needed to reduce the propagation of roundoff error during repeated arithmetic operations. The second in the emphasis on fast arithmetic operations because of the computationally-large nature of significant scientific problems. The first requirement has been met largely by increased word width. Logic technology limits have resulted in the exploration of multiprocessing designs to attack the second requirement.

A report in 1945 by Dr. John von Neumann included the concept of the general purpose, stored program computer [64]. In subsequent reports [15,32,33] co-authored by von Neumann, this concept was further refined. The structure consists of five elements: input and output mechanisms, memory for both data and instructions, arithmetic and logic unit, and control unit. The functioning of such a design is well understood today and therefore is not discussed in great detail here. In the original von Neumann design, all operations, including I-O, are done in sequential fashion with the current instruction being completed before the next instruction is started. Because of this sequential nature, it is technology that limits the performance of von Neumann machines. While the principal reason for the higher operation rate of computers is faster logic technology, and the increase in performance/cost ratio over the past two decades of computer evolution has been primarily the result of higher operation rates [8], logic speeds are reaching their upper limit [69], and hence so are SISD machines. But technology also has a secondary effect on increasing speed. More reliable devices allow large computers to be built. Smaller devices allow higher device densities. These effects allow more complex designs to be considered, designs which rely on concurrency or parallelism to achieve computing power rather than just increased logic speeds. In the remaining sections of this chapter, technology is not stressed to the extent it is here but rather emphasis will be

placed on other organizations which result in increased performance.

The objectives of overlap mechanisms include better utilization of hardware (efficiency), greater throughput (jobs per unit time), faster response (in a time sharing environment) and shorter job turnaround time (higher execution rates). Why overlap mechanisms are required to meet these needs is mainly related to the various technologies available to construct a computer. For example, although processor speeds have increased vastly as technology improved over the last few decades, the input/output facilities based largely on electro-mechanical devices have not enjoyed the same rate of speed increase. Various types of overlap have been used to meet the requirements mentioned above. Some of these are [3]:

(1) Input-output/compute overlap

(2) Interleaved memory for fetch/store overlap

(3) Instruction processing overlap (look-ahead and pipelining)

(4) Functional unit overlap (multiple arithmetic units)

(5) Pipelined functional units.

One of the first overlap methods used in computers is I-O/computation overlap. The purpose of concurrent I/O and computation is to increase resource utilization and throughput rather than to obtain a higher processor execution rate. It is not discussed further.

The second overlap mechanism is interleaved memory modules so that consecutive addresses are in different modules. Then fetching and storing of data/instructions can be performed in parallel. Two operands that are in different modules may be fetched simultaneously rather than having to wait for the rewrite cycle of the first operand to be completed before the second operand can be accessed. This mechanism can produce a higher execution rate since the memory system supplies instructions and data at a higher rate. Analyses of interleaved memory systems are contained in [37,59].

A third method of increasing SISD performance is overlapping instruction processing. In [37], Hellerman identifies a series of sub-operations required to process a single instruction. Some of these operations can be done simultaneously on different instructions. For example, one instruction may be executing while the effective address of the next instruction's operand is decoded. This type of concurrent processing requires instruction buffering or "look-ahead". Machines which use this method include the IBM Stretch [11], the first to use it, and the CDC 6600 [67,68]. Another buffering technique is "look-aside buffering memories" or "cache" memories in which both data and instructions that are frequently accessed are held. This scheme is discussed for the IBM 360/Model 85 in [8,51]. In another type of look-ahead, the term "pipeline" has been applied to a form of

operation in which a function, eg. addition, is divided into several stages each operating upon different instructions or data at the same instant. Although pipelining is discussed mainly in connection with arithmetic functional units, its use in the interpretation of the instruction stream is discussed in [8,p.84]. The MU5 computer is a recent design which uses a pipelined instruction unit [41]. Perhaps the most advanced instruction look-ahead design is exhibited by the IBM System 370/165 [45]. When a branch instruction is recognized, look-ahead occurs down both branch paths. An SISD architecture using instruction buffering and overlap is analyzed in Chapter 3.

Multiple functional units are the fourth overlap mechanism. Several specialized units operate independently of each other, and hence concurrently but under supervision. One example is the CDC 6600 [67,68] in which the main processor consists of 10 special-purpose functional units under the control of a special hardware device called the "scoreboard". It should be noted however, that the 6600 is only about 2.5 times, not 10 times, faster than the CDC 6400 - a 6600 computer but with only one shared arithmetic/logic unit [8].

A fifth method for increasing processing speeds is "pipelining" applied to the execution process, ie. pipelined arithmetic units. The IBM 360/91 [42], the first machine to implement this concept, has two floating-point units each of which are decomposed into two sub-operations. The (SISD) CDC

7600 [12] and (SIMD) CDC STAR-100 [36,40] have extended the concept.

Five types of overlap mechanisms are described above. Although presented independently, some mechanisms may enhance the utility of others and indeed some mechanisms may require the use of others. For example, efficient use of multiple arithmetic units requires some sort of instruction look-ahead or buffering. Thus, several overlap mechanisms may be found in any one machine. In fact, input-output/compute overlap and interleaved memory modules are standard features on medium to large machines in production today. While a limited amount of parallelism or concurrency is introduced by these concepts, the parallelism is still within the SISD organization and has a limited effect upon instruction execution rates.

Two types of SISD machines are recognized above -- the pure von Neumann organization operating in a strict sequential fashion and extended machines utilizing overlap mechanisms. The von Neumann machine is limited in performance mainly by the speed of the logic used to implement the design. Machines with overlap features offer increased performance but each job or task will require the same, or more, resources as it requires on a von Neumann machine. Although several types of overlap or parallelism are noted, they are all discussed as they apply to a single instruction stream operating upon a single data stream and as such are seen to offer limited improvement in the IER. This is a prime reason for exploring

the effects of concurrency at the PMS level.

2.3 MIMD Machine Organization

2.3.1 Overview

At least three types of MIMD organization can be recognized:

(1) Multiple processor computers: configurations in which several independent processors share storage (primary memory, usually) for the co-operative execution of a multi-task program [26].

(2) Computer network: a number of (usually, SISD) computers which can communicate with each other but do not share primary memory. The computers normally operate on different tasks with emphasis on sharing facilities, programs and data [7].

(3) Distributed control network [29,55]: tessellated or cellular logic configurations in which each cell is an extremely limited computer, perhaps one instruction, one data register. Cells can communicate with their neighbors in order to execute a program. Many cells can be operating upon different data at any time giving rise to the MIMD classification.

Of the above three organizations, only the first,

multiple processor computers, is dealt with in detail here. The reasons for not elaborating upon the other two organizations follow. Computer networks to date have been designed to share facilities, programs and data. In general, if a local computer cannot support a particular program, then the program and its data may be transmitted over a communication network to a larger or different type of computer. Normally, the computer network is not designed to speed the overall execution of a multi-task program by allowing parallel execution of the tasks, but it will allow parallel execution of independent programs. A computer network may be designed to handle multi-task programs (for an example, see [54]) but certain applications are not appropriate to this structure because of the delay times and limited band-width encountered in transmitting data among computers [7]. One example of a computer network is the ARPA network which connects about 25 physically remote, separate computers of various types [63]. Mini-computers are used for store-and-forward message switching and for interfacing to the larger computers. A second network example is the Distributed Computing System (DCS) built at the Irvine campus of the University of California [6,22,23]. A geographically small network of mini-computers connected by a digital communication ring, the DCS is an attempt to provide a fail-soft information utility. The use of several identical processors enables the system to continue operation with reduced performance if one of the processors fails. There is no central resource which

must function in order for the DCS to function. The possibility of parallel execution of multi-task programs seems greater in this system than in a large network like the ARPA network.

Distributed Control Networks, the third type of MIMD organization, were first proposed as a conceptual rather than a practical device to provide a formal basis for theoretical investigations in automata theory and computability. They have not been realized to any large extent and thus are not dealt with in detail here. Many organizations of this type have been proposed, the most notable perhaps the Holland machine [39] and von Neumann's cellular logic. Murtha [55] describes these and other designs briefly. Although technology is available to implement these designs, they exhibit several problems: Path interference between the many possible instruction streams, low hardware utilization factor, and, programming difficulty.

2.3.2 Multiple Processor Computers

Multiple processor computers are an evolved form of SISD organization in that they are generally thought of as general purpose problem solvers. Although the MIMD organization has the capability of solving those problems for which array processors such as the ILLIAC IV and CDC STAR-100 were built, there are many applications which do not involve the array structures required for the ILLIAC IV or STAR-100 but

nevertheless exhibit parallelism whether it is explicitly identified or implicit in the programmed algorithm [26,46,47]. Much work is being done in recognizing parallel processable streams within sequential programs written for SISD machines [4,21,34,49,57,58,61]. Some languages while designed primarily for SISD machines do allow specification of parallel tasks, eg. PL/I [43].

The objectives of applying the MIMD organization to problems are many. For computationally large problems or problems with time constraints, potential parallelism may be exploited to reduce job turnaround time. In cases where processing demands are high but involve many independent tasks or programs, the MIMD organization may be used to increase throughput. The reliability of a computer system can also be increased by using this structure. Because of processor and memory module redundancy, failure of any module will reduce but not terminate processing. As the number of jobs running or waiting to run increases, more of the system resources will be called upon at any one time than is possible in an SISD organization with only one program executing at any time. Thus the MIMD machine may exhibit high utilization factors. This can result in an increased performance/cost ratio making the MIMD organization more economical than an SISD machine with equivalent processing power.

Since many multiple processor designs are possible, a few examples of such computers are briefly described. At

Carnegie-Mellon University, a multi-mini-processor computer (C.mmp) has been designed and is now being implemented [7,9,10]. The proposed configuration consists of 16 processor systems (actually, each is a complete SISD-type computer with its own local primary memory and controllers for secondary memories and terminals) connected via a crosspoint switch to 16 primary memory modules. The processor systems are DEC PDP-11 Model 20 computers with a minimum of modifications to interface them with the rest of the design. The 16 primary memory modules each contain 2^{21} bytes. The cross-point switch has ports for m primary memory busses and n processor busses, allowing up to min(m,n) simultaneous conversations. It has no memory for port buffering, and a single processor has only one memory request pending at a time. As well, it is located centrally rather than distributed in the memory modules.

A second approach to multi-processing is taken in the Distributed Processor design [16,46,47]. It consists of a number of identical cells interconnected in a particular manner. Each cell is a general purpose processor with a small amount of memory - 512, 16-bit words. Cells are divided into groups which are connected by an intergroup bus for communication. Within each group, the cells communicate with each other by an intercell bus and by neighbor communication lines. Cells may operate in several ways. Each group has one cell designated as a controller cell. The remaining cells may be operated independently or dependently of the controller

cell giving rise to both local or global control of cells which can result in efficiently mechanizing parallel computations. That is, this MIMD organization can be configured as an SIMD organization to execute array-type operations. The design criteria included known computational requirements (storage and speed) for a space mission, high reliability and a reconfiguration time requirement after a failure in the computer system.

The last example of an MIMD organization to be given is the Shared Resource Multi-processor [25-28]. In an effort to increase utilization of system resources, this design uses pipelined execution units shared by "skeleton" processors which consist of only basic registers and a minimum of control. From a program point of view, the skeleton processor appears as a complete and independent logical computer. A combination of space-time switching is used. The time factor is the number of skeleton processors multiplexed on a time-phase ring, while the space factor is the number of multiplexed processor "rings" which simultaneously request resources.

The above three designs illustrate several ways in which processors, memories and their inter-communication system can be organized in an MIMD structure. In Chapters 3 and 4, several generalized designs are proposed, analyzed and simulated.

2.4 MIMD Parameters

As seen from the discussions of the particular computer designs above, the design process involves the architecture of three major elements:

(1) processors, P;

(2) memories, M;

(3) interface of P and M.

In the following sections these elements are discussed with regard to their respective technologies and organizations in order to determine the parameters which best describe the overall functioning of the individual elements.

2.4.1 Processor Parameters

As previously described, a major advance in electronic technology is increased circuit speed with the subsequent increase in processor speed. Such dramatic speed increases are now tapering off necessitating new processor organizations to improve processing capability [69]. One approach to increasing computation power is to increase the number of processors (Pc): if one Pc can execute i instructions per second, then ideally, n Pc can execute n times i instructions per second. However, due to switching delays, memory interference, etc., the performance of a multiprocessor is somewhat less than n times i instructions/second, resulting in a decreased performance/cost ratio. With the advent of newer

technologies, the processors may no longer be the major cost of a system [7], resulting in increased performance/cost ratio as processors are added to the system even though performance is less than the ideal maximum. The C.mmp discussed above follows this pattern [10]. Not only may the performance/cost ratio be reasonable but total system cost may now be financially feasible. These effects of processor technology are evidenced by the two recent multiprocessor projects mentioned above: the C.mmp and the Distributed Computer System (DCS), both projects of limited financial resources, relying on mini-computer class Pc but offering significant processing power. One major parameter then is the number of processors (n) in the design.

The logical organization of a Pc involves many parameters and concepts such as pipeline techniques, instruction set design, cache memory, etc. To include such detailed and varied design parameters in any one study of performance would be difficult. In this thesis, processors are characterized by the time required to execute each instruction (tei) in the instruction set, and a time (td) required to perform instruction decode and other associated functions such as instruction counter increment. These two parameters may also be used to reflect the logic technology used in implementing the processor.

2.4.2 Memory Parameters

Some elements of memory organization are word size, module size, interleaving, memory hierarchies, etc. The main parameters that are used here to describe primary memory characteristics are memory access time (ta), restore time (tw), cycle time (tc, usually equal to ta+tw), the number (m) of memory modules which can be accessed simultaneously and the capacity (bits or words) of the memory modules. For MIMD organizations, the size and number of memories becomes important since a number of processors will be competing for them. Aside from the contention that the number of memories should at least equal the number of processors , no work was found in the literature which includes memory size (number of words) as a factor in computer performance. The range of memory size in the designs considered above is large. In this study, memory capacity is not considered.

2.4.3 Processor-Memory Switching Parameters

Cost and speed are prime criteria of switching systems. Quick switching is essential because of the speeds of the elements being switched, especially if the period of connection is short and frequency of re-connection is high. The period and frequency of connections depends on the computer organization. Where switching frequency is high, switch delay time, ts, must be small or it will degrade computer performance. This delay can be taken into

consideration when analyzing system performance, cf. [10].

Efficient communication is necessary for rapid transfer of data and instructions between processors and memories in any computer system. For MIMD configurations, a major parameter is the concurrency factor, ie. the number of simultaneous conversations allowed by the switching system. A crosspoint switch is most suitable for an n-processor, m-memory architecture as it allows min(n,m) concurrent conversations. This architecture is assumed throughout the rest of the thesis.

2.5 Previous Evaluation Attempts

In this section, the methods of performance evaluation are reviewed. The two major approaches to evaluation are analytic and simulation although other approaches are sometimes used.

2.5.1 Analytic Techniques

Analytic approaches to computer evaluation fall into two classes generally - simplified queueing models and approximate solutions. The queueing models are usually based on the solution of a discrete Markov process. For this, some knowledge, real or assumed, of the job structure is required, usually in the form of a set of probabilities. Two difficulties appear with this approach, particularly for

multiprocessor systems. First, as the number of PMS components increases, the number of potential states in the system becomes quite large making it difficult to obtain other than a numerical solution. Second, it is difficult to obtain the required probabilities. References [31,52,60,65] describe examples of this approach.

Several approximate solutions have also been used. Some simplistic solutions based on the number of processors or the stage delay in a pipeline have been derived [18,19,24,26]. Some require knowledge of the job stream while others do not. A more detailed analysis of some simple SISD designs and a generalized MIMD design is done by Strecker [66]. His analysis of MIMD configurations uses basic component parameters and assumes as little knowledge of the job stream as possible.

2.5.2 Simulation

Multiprocessor systems have been simulated in efforts to determine many different parameters. Almost all of these involve simulating specific jobs. Lehman [50] simulates the solution of simultaneous equations on a generalized MIMD computer in order to determine the system's execution rate. Job scheduling policies and an estimate of executive overhead are the object of [35], while job scheduling and utilization in the ILLIAC-IV are studied in [56]. In [26-28], the job stream in a pipelined computer is simulated. The performance

of an MIMD system with multiple micro-coded processors sharing common main and control memories is simulated in [44] while "executing" existing working programs. The main drawback of these simulations is either the level of simulation (job level) or the simulation of very specific jobs. In this thesis, simulation is performed at the instruction set level with a wide variety of instruction mixes.

2.5.3 Other Approaches

One method to give an indication of execution rate in a mono-programming computer is to count the instructions that would be executed for a job written in the computer's machine code. This was done to compare the ILLIAC-IV and CDC STAR-100 computers [36]. The same problem was first coded in each machines assembly language and then using projected instruction times an estimate of each job's duration was obtained. This approach is difficult to apply to a multi-programming, multiprocessing computer due to the large number of interacting parameters. Another approach taken is to divide jobs into tasks to determine how much parallelism is required in the system. Koczela [46,47] follows this approach in the design of a specialized aerospace MIMD computer. Dynamic programming is used to optimize the cost/performance ratio over a given set of system parameters in [31].

2.6 Summary

In this Chapter, four machine organizations--SISD, SIMD, MISD and MIMD--were identified according to the number of instruction and data streams which they are intended to handle. Of these four, the SISD and MIMD organizations were reviewed to provide a basis for further research. The high level components which compose multiprocessors were isolated and the parameters which describe their individual performances identified. It is in terms of these parameters that multiprocessor performance is analyzed in Chapter 3 and simulated in Chapter 4. A brief review of previous attempts of multiprocessor evaluation was also given.

CHAPTER 3

ANALYTIC MODELS OF CERTAIN MIMD DESIGNS

3.1 Introduction

In Chapter 2, the major components of a computer system were studied in an effort to describe their respective roles and importance in the overall architecture of the computer. Also, an attempt was made to determine those parameters which best describe the functioning of the individual components, e.g. memory access and cycle times and processor execution times. In this Chapter some computer designs are described in terms of these parameters in order to determine their "performances". Only one performance parameter, the instruction execution rate (IER), is studied in this Chapter. The IER and other performance parameters are further discussed in Chapter 4 where the analytic results of this chapter are compared to simulation results of the same systems. The analysis does not consider the specific job stream which might be processed on the computer. Rather it is assumed that jobs exist which can be broken down into parallel-executable tasks or processes sufficient to keep the processors occupied. It is also assumed that memory accesses are uniformly and randomly distributed throughout the set of memories in the system.

This approach is taken for the following reasons:

(1) It is difficult to obtain good job parameter descriptions for existing jobs over a wide range of applications [66].

(2) The level of analysis here (PMS level) is too fine to be compared to an approach which uses gross job parameter values, e.g. inter-arrival times, job priority, etc.

(3) Progress is being made on the recognition of parallelism at all levels [17,21,34,49,57], so the assumption that processes will be available to occupy MIMD machines seems valid.

(4) If a multiprocessor is not taxed by one job running in mono-programming mode, it can be multiprogrammed for better resource utilization. In a multiprogramming mode, the task of determining job parameters becomes even more difficult. Also, multiprogramming tends to randomize memory accesses.

A generalized configuration of an MIMD organization in which programs and their data are separated and stored in different memories (disjoint memories) is shown in Figure 3.1. The concept of separating instructions from their data to produce "pure procedures" has been implemented on SISD computers for some time [30]. The impetus for doing this seems to be the desire to reduce code duplication and allow code sharing rather than primarily to increase computer performance. In the following sections of this chapter, the effect of utilizing separate program memories (Mp) and data memories (Md) on multiprocessor performance is analyzed.

Although certain assumptions and restrictions are made in this thesis, the generality of the design in Figure 3.1 should not be over-looked. For example, although two switches (S) are shown (logically) in the diagram, their realization would most likely be one switch. Also, although fixed values of m and k, the number of data and program memories, respectively, are used in the analyses here, it is desirable that m and k would be varied either manually or, preferably, dynamically as the processing demands change in a real system subject to the restriction m+k=constant. It is even possible that some of the instruction memories be of the read-only type in which infrequently changed system programs reside, eg. compilers, editors, even the operating system itself.

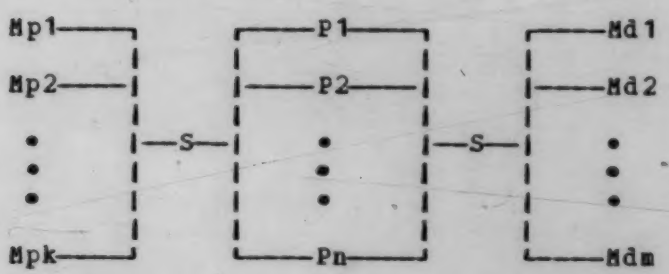
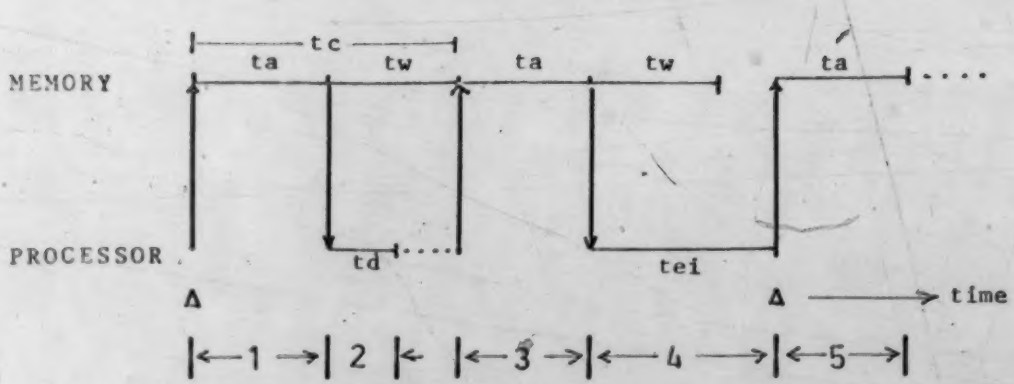


Figure 3.1. MIMD Design with Disjoint Memories

3.2 Instruction Timing Diagram (ITD)

Throughout the chapter, it is assumed that single address format instructions [8] are being executed. This format is implemented on a large number of computers and is one of the most frequently executed instruction types [71] although several formats usually exist in a computer. The single address format requires two memory references: one for the instruction itself and one for the operand. It is assumed that an instruction and an operand each occupy exactly one memory word. The execution of a single address instruction is represented by the Instruction Timing Diagram (ITD) in Figure 3.2. The actual execution of an instruction is more complex than is represented by the figure but some operations in a real computer, eg. incrementing of instruction counter, are performed in parallel with the events shown in the diagram and are thus transparent to the analysis.

If more than one memory is capable of being accessed simultaneously, delays in the system may be reduced by directing successive memory requests to different memories. For example, if instructions and their operands are stored in different memories, the delay after the instruction decode in Figure 3.2 may be eliminated thus increasing the rate of instruction executions. The concept is further developed in succeeding sections.



Legend:

1 instruction fetch
2 instruction decode
3 operand fetch
4 instruction execution
5 next instruction fetch

ta memory access time
tw memory restore time
td instruction decode time
tei processor execution time
Δ-Δ time for complete instruction execution
tc memory cycle time

Figure 3.2. Instruction Timing Diagram (ITD)

3.3 SISD Computer with Buffering

Strecker [66] analyzed SISD computers with overlap mechanisms to illustrate the limitations of SISD design. Since more complex SISD designs than those studied by Strecker have been developed, one such SISD design is analyzed to explore the possible improvement in performance.

This design (Figure 3.3) is an SISD computer with instruction buffering and instruction prefetch, i.e. overlapped or pipelined instruction decoding and overlap of execution and instruction prefetch/decoding. The analysis does not consider overlapping or pipelining of the instruction execution phase. The main memory is assumed to be m-way interleaved. A study of the ITD for this design, Figure 3.4, reveals the cases shown in Figure 3.5.

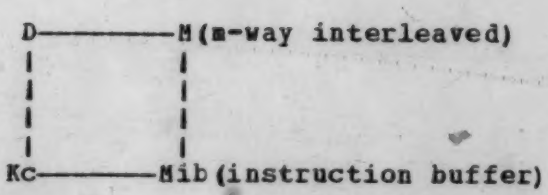
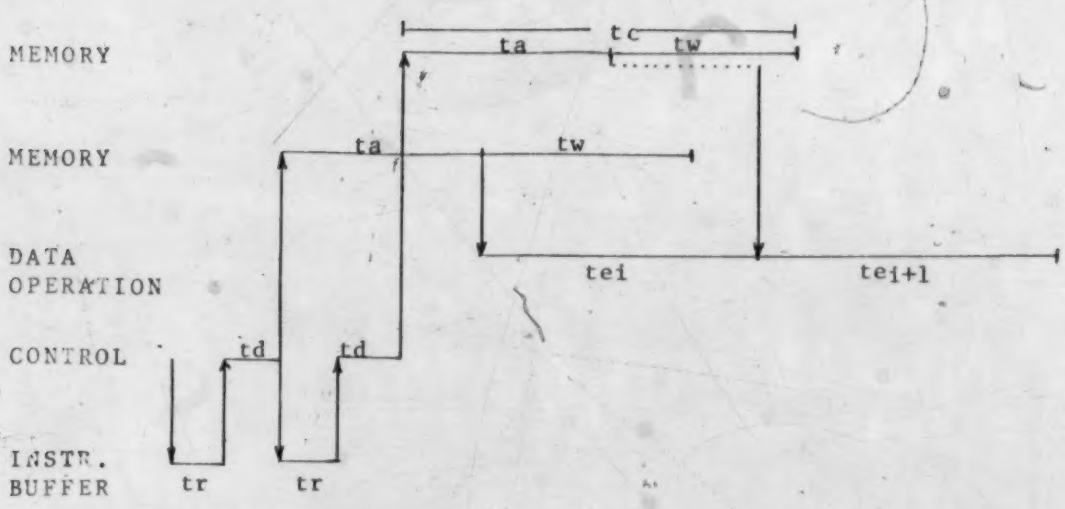


Figure 3.3. SISD design with Buffering



Legend: as for Figure 3.2.
tr instruction buffer access time

Figure 3.4. ITD for SISD computer with overlap.

Class	Conditions	Execution Time (ti)
1a	tei≤tw; td+tr<tei	ti=tc/m + (1-1/m)*tei
1b	tei≤tw; td+tr≥tei	ti=tc/m + (1-1/m)*(td+tr)
2	tw < tei < tc	ti=tc/m + (1-1/m)*tei
3	tei ≥ tc	ti=tei

Figure 3.5. Execution Time for ITD of Figure 3.4 (Factors 1/m and (1 - 1/m) indicate presence or absence, respectively, of memory access conflict; tr + td << ta,tw)

The expected instruction execution time, E(t), (without branches) is calculated by determining

$$E'(t) = \sum fi*ti$$

where fi is the frequency of execution of instruction i and ti is the execution time of instruction i. This summation can be written as

$$E'(t) = \sum_{i\epsilon C1a} fi*ti + \sum_{i\epsilon C1b} fi*ti + \sum_{i\epsilon C2} fi*ti + \sum_{i\epsilon C3} fi*ti \quad (3.1)$$

where $i\epsilon C1a$, etc. mean those subscripts which apply to Class 1a, etc. Define:

$$f1a = \sum_{i\epsilon C1a} fi$$

and $$te1a = (1/f1a) * \sum_{i\epsilon C1a} fi*tei.$$

From these definitions f1a is the relative frequency of all instructions of Class 1a and te1a is the average processor execution time for all instructions of Class 1a. Similar definitions, not shown, are made for Classes 1b, 2 and 3. Then, from equation 3.1 and Figure 3.5,

$$\begin{aligned} E'(t) = & \sum_{i\epsilon C1a} fi*(tc/m+(1-1/m)*tei) \\ & + \sum_{i\epsilon C1b} fi*(tc/m+(1-1/m)*(td+tr)) \\ & + \sum_{i\epsilon C2} fi*(tc/m+(1-1/m)*tei) \\ & + \sum_{i\epsilon C3} fi*tei \\ = & (tc/m)*(\sum_{i\epsilon C1a} fi + \sum_{i\epsilon C1b} fi + \sum_{i\epsilon C2} fi + \sum_{i\epsilon C3} fi) \\ & + (1-1/m)*(\sum_{i\epsilon C1a} fi*tei + \sum_{i\epsilon C1b} fi*tei + \sum_{i\epsilon C2} fi*tei + \sum_{i\epsilon C3} ti*tei) \end{aligned}$$

$$+ (1-1/m)*[(td+tr)*\sum_{i\in C1b} fi + \sum_{i\in C1b} fi*tei]$$

$$+ \sum_{i\in C3} fi*tei - (1-1/m)*\sum_{i\in C3} fi*tei - (tc/m)*\sum_{i\in C3} fi$$

$$= tc/m + (1-1/m)*tem + f1b*(1-1/m)*(td+tr-te1b)$$

$$+ (f3/m)*(te3-tc). \qquad (3.2)$$

where tem is the average processor execution time for all instructions:

$$tem = f1a*te1a + f1b*te1b + f2*te2 + f3*te3.$$

In equation 3.2, the first term is the delay caused by branches in the instruction buffer. The fraction (1-1/m) is the probability that an accessing conflict will not occur between instructions. All the instructions are represented by tem, the average processor execution time, in the second term but Classes 1b and 3 produce residual effects in the observed mean execution time as shown in the third and fourth terms of equation 3.2.

Equation 3.2 simplifies under certain conditions. For a large number of memories (m), the first and last terms tend to zero while the second and third terms go to tem and f1b*(td+tr-te1b), respectively. For a processor with a fast instruction buffer (tr small), the third term also drops out leaving E(t)=tem, the mean processor execution time. This is an expected result under the assumptions made. The mean IER is just the reciprocal of the mean execution time, i.e. IER goes to 1/tem. This represents an ideal condition with no delay due to accessing conflicts nor does it consider the

possibility of branch instructions in the instruction stream.

The existence of a branch instruction means that less than the maximum number of instructions in the instruction buffer will be executed before the instruction buffer must be refreshed. Assume that each time a branch is recognized, the instruction buffer is completely refilled. Also, assume it contains m instructions where m equals the number of interleaved memories. Then the time to fill the instruction buffer is tc, the cycle time, because the first operand reference conflicts with an instruction fetch. To compute E(t), the expected execution time for a single instruction, the initial instruction fetch time is apportioned among $s < m$ instruction executions, i.e.

$$E(t) = tc/s + E'(t)$$

where E'(t) is the expected execution time without buffer refill or branch, that is, the E(t) derived in equation 3.2.

Under the assumptions that each branch is out of the instruction buffer (Mib), that no look-ahead down each branch path takes place, and that there is a constant probability, Pb, that any instruction is a branch, it has been shown in [66] that the expected number of instructions executed before a branch occurs is

$$E(x) = [1-(1-Pb)^{m}]/Pb. \qquad (3.3)$$

This value can be used as an approximation for s in the equation for E(t) immediately above. Therefore,

$$IER = 1/E(t) = 1/[tc/s + E'(t)] =$$

$$\frac{1}{\frac{tc * Pb}{(1-(1-Pb)^m)} + \frac{tc}{m} + (1-\frac{1}{m})\,tem + f1b\,(1-\frac{1}{m})\,(td+tr-te1b) + \frac{f3}{m}(te3-tc)} \quad . \quad (3.4)$$

The last four terms in the denominator are those arising from the determination of E'(t) (equation 3.2) and were explained above. The first term is new and represents the delay in instruction execution due to branches occuring with probability Pb. (Recall that the instruction buffer is refilled when a branch outside of it takes place.) Similar to equation 3.2, equation 3.4 reduces under certain conditions, viz. a fast access instruction buffer (small tr), a large number of memory modules (m) and reasonable probability (Pb) of a branch occurring, to:

$$IER = 1/(tc*Pb + tem). \quad (3.5)$$

The effect of the branch instruction on the performance of this SISD system is readily apparent. If Pb is small, say .05, then the execution rate depends mainly on the speed of the processor (tem). Conversely, a fast processor (small tem) is only worthwhile if branching is kept to a minimum.

In general, equation 3.4 shows that the IER is a balance of the main memory speed, processor speed and the instruction mix (degree of branching). This result shows that even with a large degree of overlap, SISD performance is still mainly limited by logic technology, in particular, memory technology (tc) and processor execution technology (tem). For a

uniprocessor with one memory (m=1), the term (1-1/m) is zero in equation 3.4 reducing it directly to

$$IER = 1/(2tc+f3(te3-tc)).$$

This shows that the execution rate (IER) in a uniprocessor is primarily a function of main memory speed (tc). The same result is obtained in [66] for a uniprocessor without overlap and with f3, the frequency of instructions with long execution times, equal to zero.

3.4 Multiple SISD Computers with Disjoint Memories

To create a base for comparison, this section analyses the effect of separate program memories and data memories on the performance of a uniprocessor. Referring to Figure 3.1, if each processor is restricted to accessing instructions from a specific program memory and operands from a specific data memory , with no interference from other processors , then the generalized MIMD design becomes n independent SISD computers. Figure 3.6 shows the Instruction Timing Diagram (ITD) for this design.

In this and subsequent analyses of the IER, the value tp = fi*tei which is the average processor execution time over the instruction set, will be used to reflect the "speed" of the processor. Henceforth, it is assumed that all instructions of the instruction set have an execution time of tp.

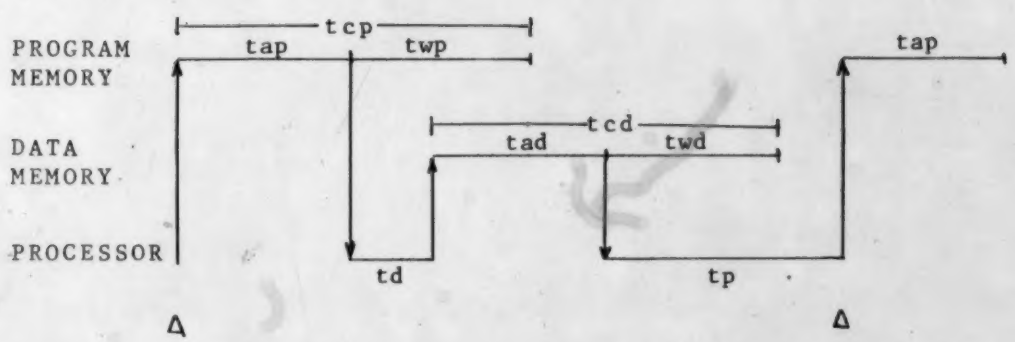
Assuming (a) td+tad+tp≥twp (no conflict at Mp) and (b) tp+tap+td≥twd (no conflict at Md), the execution rate for one SISD computer is readily derived from the ITD of Figure 3.6:

$$IER = 1/(tap + td + tad + tp).$$

For an n processor system, the IER would be n times this. A similar result for the SISD computer with an m-way interleaved memory, without conflicts, and large m is derived in [66]. In the uniprocessor case, the division of instructions and data into separate memories presents a

possible alternative to the interleaving of memory in a system where instructions and data are intermixed.

For an arbitrarily fast processor, tp --> 0, conditions (a) and (b) above indicate that the read and restore times of the memories should be approximately equal. On the other hand, for a slow processor, the IER can be improved by using fast access, slow restore memories.



Legend: tcp program memory cycle time
tap program memory access time
twp program memory restore time
tp mean processor execution time

Figure 3.6. ITD for SISD Computer with Disjoint Memories

3.5 MIMD with Dedicated Program Memories

A dedicated program memory multiprocessor design is shown in Figure 3.7.

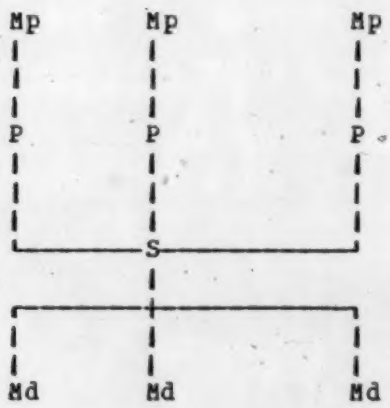
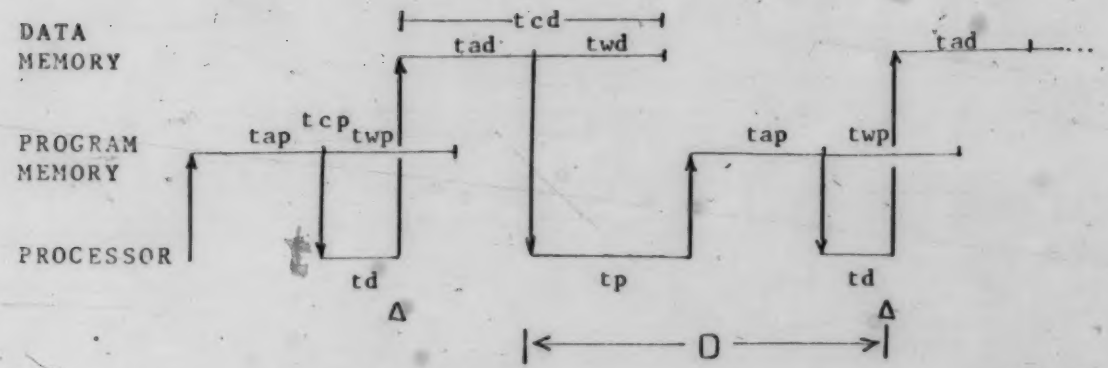


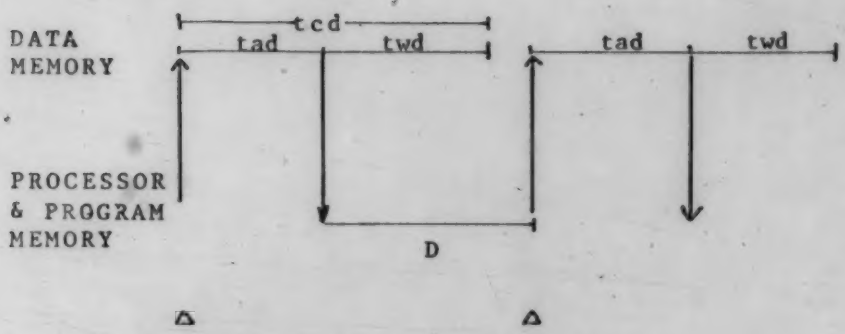
Figure 3.7. MIMD with Dedicated Instruction Memories

Such a configuration could arise where, say, a compiler executing in one program memory/processor pair processes several different data sets (source programs) stored in different data memories. From the ITD (Figure 3.8), it is seen that a memory access conflict at the program memories arises only if td+tad+tp < twp. To prevent conflicts at the program memories, then, the condition td+tad+tp ≥ twp must be true. This conditon is easily met in a design which is dedicated to specific applications since the instruction memories can be made to be read-only, i.e. twp=0. The possibility of conflicts at the program memories is not considered in the following analysis.



Legend: as for Figure 3.6

Figure 3.8. ITD for MIMD Computer with Dedicated Program Memories



Legend: as for Figure 3.6

Figure 3.9. Simplified ITD for Figure 3.8

Since the average processor execution time (tp) is treated as a constant, the execution of an instruction in the above system may be thought of as a data memory access followed by a delay, D = tp+tap+td, before the next instruction is executed. The above ITD can be redrawn as in Figure 3.9. This conceptual simplification, a random data access from one of m data memories followed a fixed delay, results in a system similar to the one solved in [66] but with the following difference. Instead of using "unit instructions" (one memory access followed by a delay equal to (tp+td)/2), the ITD above represents "real" instructions. Three cases arise from the above ITD: D=twd; D>twd; and D<twd. The IER for each of these, as derived in [66], are shown in Figure 3.10. The term $[1-(1-1/m)^n]$ represents the effect of n processors being queued on m data memories. (All n processors are queued because D≤twd, as shown in Figure 3.9.) The parameter Pm is the probability that a processor is queued on one of the data memories. It is a function of the number of processors (n) and data memories (m). These analytic models are compared to simulation results in Chapter 4.

1. D=twd: $IER=(m/tcd)*[1-(1-1/m)^{n}]$

2. D<twd: $$IER=(m/tcd)*\frac{[1-(1-1/m)^{n}]}{1-[(1-1/m)^{n}]*[(twd-D)/tcd]}$$

3. D>twd: $IER=(m/tcd)[1-(1-Pm/m)^{n}]$

where $Pm=1-(m/tcd)[1-(1-Pm/m)^{n}]*[(D-twd)/n]$

Figure 3.10. Instruction Execution Rates for MIMD with Dedicated Instruction Memories (D=tp+tap+td; td+tad+tp≥twp)

3.6 Generalized MIMD with Disjoint Memories

The multiprocessor design considered in this section has a set of m operand memories, a set of k program memories and a set of n independent processors (Figure 3.1). Instruction execution proceeds in the following manner. A processor chooses a program memory (Mp) obtains an instruction from it and decodes that instruction. Assuming a single address format instruction an operand is required so the processor then chooses a data memory (Md) and upon fetching the desired operand, completes the execution of the instruction. The processor then repeats this cycle. In this study, the accesses to the k program memories and m data memories are assumed to be randomly distributed. This system may be thought of as a feedback queueing system with two sets of servers, the k program memories and m data memories, servicing the requests of the n processors for memory cycles (Figure 3.11). Its ITD is shown in Figure 3.12.

From the instruction format it is observed that one instruction is executed for each program memory cycle which is followed by a data memory cycle. The IER, then, depends on the rate at which memory cycles become available, the maximumum rate being k/tcp for the instruction memories and m/tcp for the data memories. An upper bound on the IER is min[k/tcp,m/tcp]. In this section, the problem of finding the IER is analyzed by determining that fraction of the memory cycles which are utilized by the n processors.

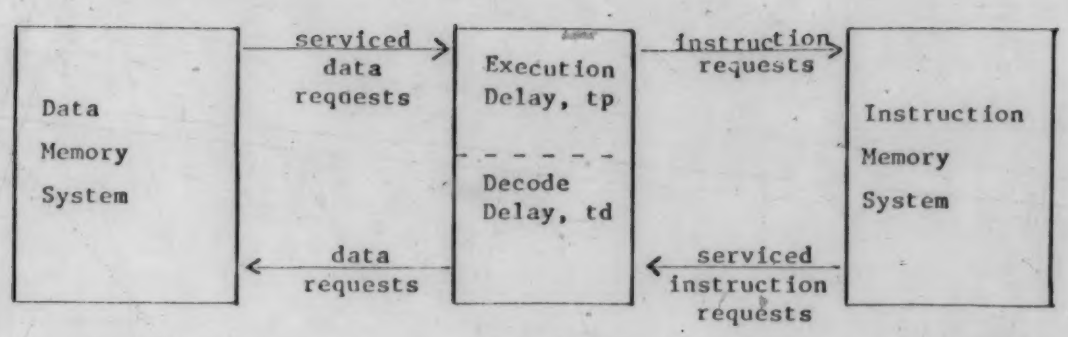
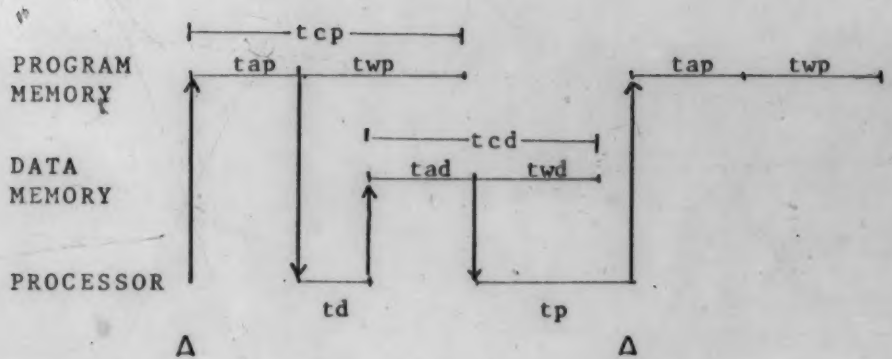


Figure 3.11. Instruction Execution Flow in MIMD Computer with Disjoint Memories



Legend: as for Figure3.6

Figure 3.12. ITD for MIMD Computer with Disjoint Memories

A processor is said to be queued if it is either waiting for or receiving memory service. It may be queued on either a program memory or a data memory . A memory is termed occupied if it has one or more processors queued and unoccupied if it has no processors queued. A processor is termed active if it is currently being served by a memory, either a program memory or a data memory.

Consider the instruction memory system. Assume a constant probability, Pp, that a given processor is queued for memory service on the k program memories. Then 1-Pp is the probability that it is not queued on the k program memories. The probability that v out of n processors are queued is given by the binomial distribution:

$$P(v) = \binom{n}{v} (Pp)^{v} (1 - Pp)^{n-v} \qquad (3.11)$$

When v processors out of n are queued, the rate at which instruction fetches are serviced is given by RP(v)=(average number of occupied program memories)/tcp. Since it is assumed that the v processors have randomly distributed their requests among the k program memories, the problem of determining the average number of occupied program memories can be treated by combinatorial analysis, and has been shown [62] to equal

$$g(k,v) = k*(1-(1-1/k)^{v}). \qquad (3.12)$$

Therefore,

$$RP(v) = k*(1-(1-1/k)^{v})/tcp. \qquad (3.13)$$

The expected value of RP(v), E(RP), is

$$E(RP) = \sum RP(v)*P(v)$$

$$= \sum (k/tcp)*(1-(1-1/k)^{v}\binom{n}{v}(Pp)^{v}(1-Pp)^{n-v}$$

$$= (k/tcp)*(1-(1-Pp/k)^{n}). \qquad (3.14)$$

(The latter by application of the binomial theorem.)

The expected rate of instruction fetches, E(RP), is a function of Pp, the probability that a processor is queued on the instruction memory system. Before developing an expression for Pp, a similar expression for the rate of operand fetches from the data memory system is developed.

If Pd, a constant, is the probability that a given processor is queued on the m data memories, then the probability that u out of n processors are queued is given by

$$P(u) = \binom{n}{u}(Pd)^{n}(1 - Pd)^{n-u} \qquad (3.15)$$

and the rate at which operand requests are serviced is given by RD(u)=(average number of occupied data memories)/tcd, ie.

$$RD(u)=m*(1-(1-1/m)^{u})/tcd. \qquad (3.16)$$

The expected value of RD(u), E(RD), is derived in the same fashion as E(RP):

$$E(RD)=m*(1-(1-Pd/m)^{n})/tcd. \qquad (3.17)$$

The non-zero values of tp and td do not affect in a direct way the rates of instruction and data fetches; however, their effects are felt indirectly through their influence on the values of Pd and Pp.

To derive an expression for Pp, it is noted that since the number of processors queued at the k program memories is binomially distributed, the average number of processors queued is n*Pp. Therefore,

Pp=(average number of P queued at k Mp)/n. (3.18)

Serviced instruction fetch requests leave the instruction memory system at a rate specified by equation 3.14. They then experience a delay, t1, before again making an instruction fetch request. Let Nn be the average number of n processors not queued at the k program memories and Nq be the average number queued. Then Nq=n-Nn, which substituted into equation 3.18 gives

$$Pp = (n - Nn)/n = 1 - Nn/n. \quad (3.19)$$

The average number of processors not queued at the k program memories must be the product of the average delay, E(t1), and the average instruction fetch rate:

$$Nn = E(RP)\,(Pp)*E(t1) \quad (3.20)$$

Hence:

$$Pp = 1 - (E(RP)*E(t1))/n$$

$$= 1-(k/ntcp)[\,1-(1-Pp/k)^{n}\,]*E(t1). \quad (3.21)$$

The value of E(t1) equals the sum of the instruction decode time, td, plus the expected time to fetch an operand, plus the

instruction execution time, tp. The time to fetch an operand depends on whether the processor accesses an occupied or unoccupied memory, ie. whether it is delayed or not. The probability, P(occ), that a processor accesses an occupied data memory when u processors are queued at the m data memories is

$$P(occ) = (1-(1-1/m)^{u}) \tag{3.22}$$

and the probability of the processor accessing an unoccupied memory is

$$P(unocc) = 1 - P(occ) = (1-1/m)^{u}. \tag{3.23}$$

Thus the delay, t1, when u processors are queued is:

$$\begin{aligned} t1(u) &= td + \text{expected time to fetch operand} + tp \\ &= td + P(occ)*tcd + P(unocc)*tad + tp \\ &= td + [1-(1-1/m)^{u}]*tcd + [(1-1/m)^{u}]*tad + tp \\ &= td + tp + tcd - [(1-1/m)^{u}]*twd. \end{aligned} \tag{3.24}$$

The expected value of t1, E(t1), is:

$$\begin{aligned} E(t1) &= \sum t1(u)*P(u) \\ &= \sum [td+tp+tcd-((1-1/m)^{u})*twd]\binom{n}{u}(Pd)^{u}(1-Pd)^{n-u} \\ &= td+tp+tcd-twd*(1-Pd/m)^{n}. \end{aligned} \tag{3.25}$$

This value is inserted into equation 3.21 to evaluate Pp. Using analagous arguments, an expression for Pd is derived. The equations describing the system are summarized below.

$$E(RP) = (k/tcp)[1-(1-Pp/k)^{n}] \quad (3.26)$$

$$E(RD) = (m/tcd)[1-(1-Pd/m)^{n}] \quad (3.27)$$

$$Pp = 1-(k/ntcp)[1-(1-Pp/k)^{n}][td+tp+tcd-twd*(1-Pd/m)^{n}] \quad (3.28)$$

$$Pd = 1-(m/ntcd)[1-(1-Pd/m)^{n}][td+tp+tcd-twd*(1-Pp/k)^{n}] \quad (3.29)$$

Equations 3.28 and 3.29 are solved for Pp and Pd in the interval (0,1). That there exists one and only one solution for the two equations in (0,1) is seen by considering equation 3.28. Holding Pd constant, as Pp goes from 0 to 1, the left hand side increases monotonically from 0 to 1 while the right hand side decreases monotonically from one. Thus for a given value of Pd in (0,1) there is one and only one value of Pp in (0,1) for which the right hand side equals the left hand side. An iterative procedure which alternately solves equations 3.28 and 3.29 leads to the correct values of Pp and Pd. Once obtained, they are substituted into equations 3.26 and 3.27 to determine the rate of instruction and operand fetches. The IER is the minimum of (E(RP),E(RD)). For large numbers of processors (n), E(RP) tends to k/tcp, the number of program memories divided by the memory cycle time, and E(RD) tends to m/tcd which agrees with the maximum rates mentioned at the start of this section. In Chapter 4, the above system of equations is solved for various configurations.

3.7 Summary

In this chapter several computer designs were analyzed. These designs are:

1. SISD computer with buffering.
2. Multiple SISD computers with disjoint memories.
3. MIMD computer with dedicated program memories.
4. MIMD computer with disjoint memories.

Mathematical models were derived for the execution rate of these systems. In the analyses, two assumptions were made. The first was that all instructions in the system are of the same class, the single address format. The second assumption was the treating of the inherent multiprocessing queueing problem as a distribution or occupancy problem.

CHAPTER 4

SIMULATION AND RESULTS

4.1 Introduction

In this chapter, the results of two sets of simulation experiments are reported. The objectives of the experiments are the study of the two performance parameters -- instruction execution rate (IER) and resource utilization. In the first set of simulations, the instruction execution rate is studied by a series of experiments. In the first series, the analytic results of Chapter 3 are compared to simulation results using the single address format instruction. The second series, also using the single address format, compares, through simulation, different computer designs--MIMD with mixed, disjoint and dedicated memories. A third series describes the effect of using only the single address instruction by comparing its results to experiments run using a variety of instruction mixes.

The second set of experiments was performed to study resource utilization. Two series were run, one to compare the utilization of the different MIMD designs and one to study the effect of instruction mix on utilization. The chapter concludes with a brief summary of the results obtained in the series of experiments.

4.2 The Simulation Model

The simulator is the "next most imminent event" type in which rules are applied to determine the sequence of events in the simulated system and the timing of events is determined accordingly. The simulator can handle n processors denoted Pi, i=1,2,...,n; m data memories denoted MDj, j=1,2,...,m; and k program memories denoted MPr, r=1,2,...,k where k,m,n are arbitrary input parameters to the simulator. One cycle of simulation corresponds to either an instruction fetch from one of the k MP followed by the instruction decode, or an operand fetch from one of the m MD followed by the instruction execution phase. Therefore two cycles are required to "execute" one instruction of the single address format. The sequence of events in the simulator (Figure 4.1) is governed by three rules. The instruction fetch or operand fetch of any cycle is associated with the processor Pi which has the minimum elapsed time at the start of the simulation cycle. Also, an instruction fetch/decode sequence involving Pi and MPr commences at max[TPi,TMPr], where TPi and TMPr are the elapsed times for Pi and MPr, respectively since that is the earliest time at which both Pi and MPr are available. Similarly, an operand fetch/execution sequence involving Pi and MDj commences at max[TPi,TMDj].

The input parameters to the simulator are precisely those used to describe the computer systems in Chapter 3, ie. memory access and restore times, processor decode and

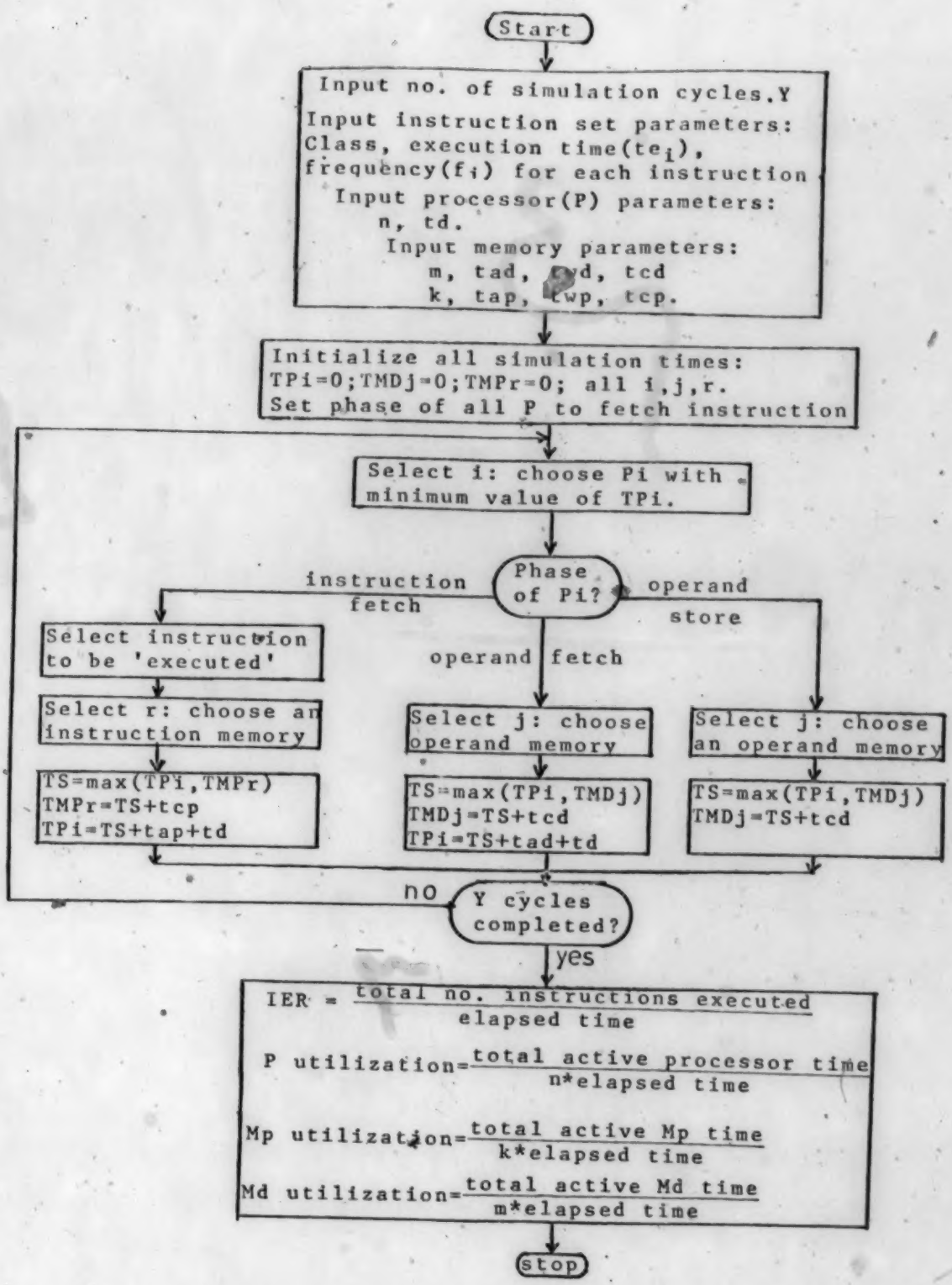


Figure 4.1. Simulator Flow Diagram

execution times and the number of elements of each type in the system. To reflect the different designs studied, several versions of the simulator were programmed. Figure 4.1 reflects the disjoint memory design. Only minor changes are required to reflect the mixed memory and the dedicated program memory architectures. The simulator has provision for several instruction formats and accepts a description of the instruction set with parameters for format type, execution time and relative frequency of each instruction. This is discussed in more detail in Section 4.3.3.

Instructions are "executed" randomly, that is, a pseudo-random number generator is used to determine which instruction in the instruction set is to be executed next. The memory module to be accessed is also determined randomly except in the dedicated memory case. The seed of the random number generator is kept constant so that the same instruction sequence is used to test the different configurations. Tests were also made using different seeds for the random number generator to determine if any of them produced anomalous results. None were observed for the seeds chosen. The number of cycles simulated may also affect the results observed. For small configurations with less than eight components of each type, the simulation results for 10,000 and 30,000 cycles differed only in the third or fourth significant digit. For larger configurations, the larger number of cycles was required to obtain consistent results when testing different

random number generator seeds.

The output of the simulator consists of a list of input parameters, detailed statistics and summary statistics of the simulated execution. The detailed statistics include the number of instructions executed, the simulated elapsed time of execution, the IER and the utilization for each processor, and the number of accesses, elapsed time and utilization for each memory module. The summary statistics include the simulated system's averages for those parameters mentioned above for individual components. The simulator is written in FORTRAN and when executing on an IBM 360/67, simulates the execution of 1000 single address instructions in about 0.4 seconds.

4.3 Instruction Execution Rate Results

In this section the results of the simulation and analytical models are reported for the two series of tests. The first series studies the instruction execution rates of the designs while resource utilization is the object of the second series. In the first series, in order to compare the designs, the instruction execution rates are plotted relative to the execution rate of a one processor, one memory computer executing a single address instruction. A moderate speed processor is used with the average processor execution time (tp) equal to the memory restore time (tw).

4.3.1 Analytic and Simulation Results

In the previous chapter, the inherent multiprocessing queueing problem was treated as an occupancy or distribution problem. This section studies the effect of this approximation over a limited set of cases. Two series of results are presented: the first deals with the MIMD design with disjoint memories, the second with the MIMD design with dedicated instruction memories.

Figure 4.2a shows the instruction execution rates of the analytic model and the simulation model for the MIMD design with disjoint memories. In these results the instruction memory speed is equal to the data memory speed and a medium speed processor with tp=tw is used. The simulation produces

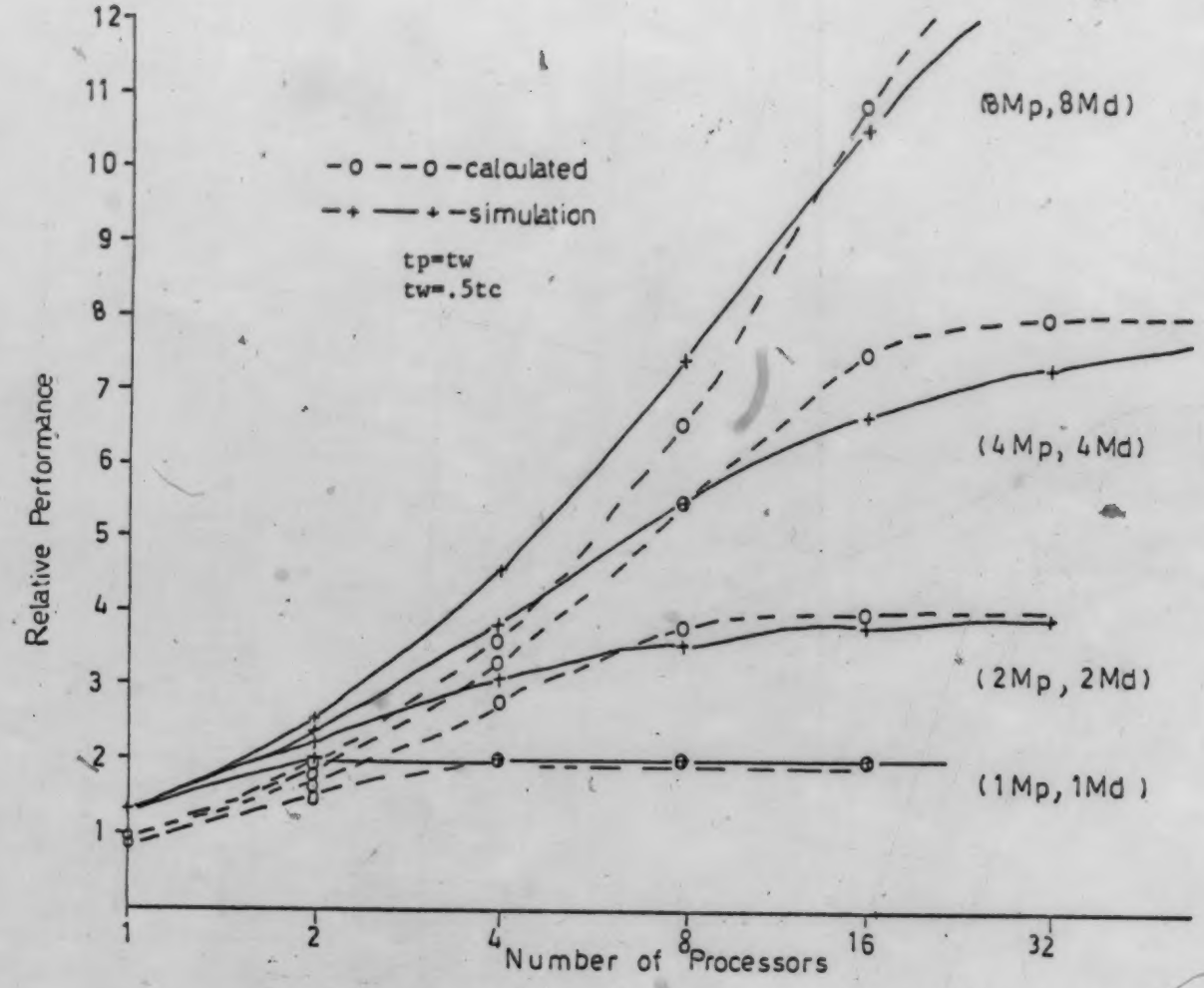


Figure 4.2a.MIMD Performance with Disioint Memories

results which compare favorably to the analytic model. The differences are due to the linearizing assumptions made in the analytic model and the procedural assumptions in the simulation model. The execution rate reaches the same asymptotic value for both models for large numbers of processors. In all cases, as processors are added to the system, the relative performance reaches a maximum value equal to the number of memories in the system. This value is observed because the relative execution rate is being plotted. In the limit, the performance of the system is determined by the number of memories. Thus a multiprocessor with m memories has an execution rate m times that of a single memory computer. This also agrees with the observation made in Chapter 3 that the maximum execution rate equals min[k/tcp,m/tcd].

The models were also compared using fast and slow processors with tp=.5tw and 2tw, respectively. The maximum difference between the models rose slightly, about 5%, for the fast processor and decreased about the same for the slow processor. Thus the disagreement between the models is mainly a result of memory characteristics and number of processors, not processor speed. It is seen that the analytic model always produces execution rates lower than the simulation for small values of n, the number of processors. The analytic model also reaches the asymptotic performance level before the simulation model, the only exception observed is for the one

program memory (1Mp), one data memory (1Md) case. In the analysis of the MIMD computer with disjoint memories, a number of processors, u and v, were assumed queued at the operand and instruction memories and the expected number of occupied memories was estimated by equation 3.12. In a real computer $u' \leq u$ and $v' \leq v$ processors are probably queued since if there are several processors queued for service at a memory, only the one serviced during the memory cycle makes a new request at the end of the cycle. Thus the expected delay through the memory systems is somewhat less than predicted by equation 3.25 and the execution rate in a real computer may be higher than predicted by equations 3.26-3.29. The two models appear to have no significant anomalies. Each produces a family of performance curves that exhibit the same characteristics.

Results for the MIMD machine with dedicated instruction memories are shown in Figure 4.2b. Here it is seen that the analytic model closely reflects the simulation model, the maximum difference being about 10% for the cases studied. Generally the analytic results are higher than the simulation results with the maximum differences occuring just before the asymptotic performance levels are reached. In this region, the ratio of processors to memories is high. When this occurs, the possibility of unfavorable queuing conditions increases resulting in decreased execution rates. The simulation model can reflect this condition whereas the analytic model, using statistical averages, can not. As in

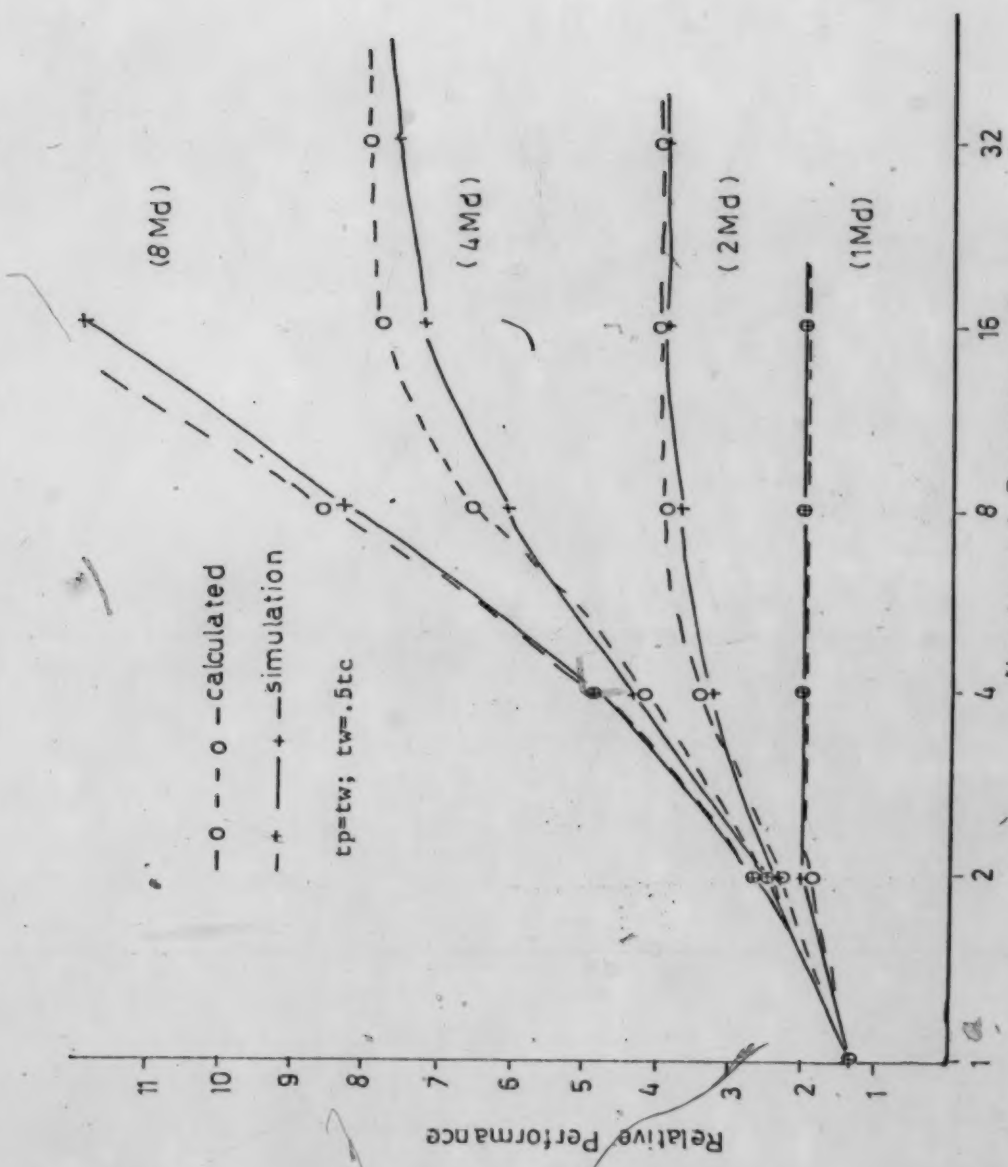


Figure 4.2b MIMD Performance with Dedicated Program Memories

the disjoint memory case above; the dedicated program memory models were tested using faster and slower processors. For a four-fold change in processor speed, from tp=.5tw to tp=2tw, the maximum difference between the models changed negligibly remaining near 10%.

One reason for the better agreement observed between the dedicated program memory models than for the disjoint memory models is the degree of complexity in the two designs. In the dedicated memory design, queueing occurs only at one set of memories while in the disjoint design it occurs at both sets. Thus the differences between the simulation and the analytic models will be compounded in the disjoint design. In order to decrease these differences, a more comprehensive analytic model is required.

4.3.2 Simulation Results using Single Address Format

In this section, the simulation results for the instruction execution rate of the three MIMD designs operating under the same instruction mix, single address format, are reported. The three designs are MIMD with mixed memories, disjoint memories and dedicated instruction memories.

Figures 4.3a,b compare the mixed memory system to the disjoint memory system using tp=0 and tw, respectively. Figure 4.3a shows only the effects of the different memory designs since the processor speed is zero. Figure 4.3b shows

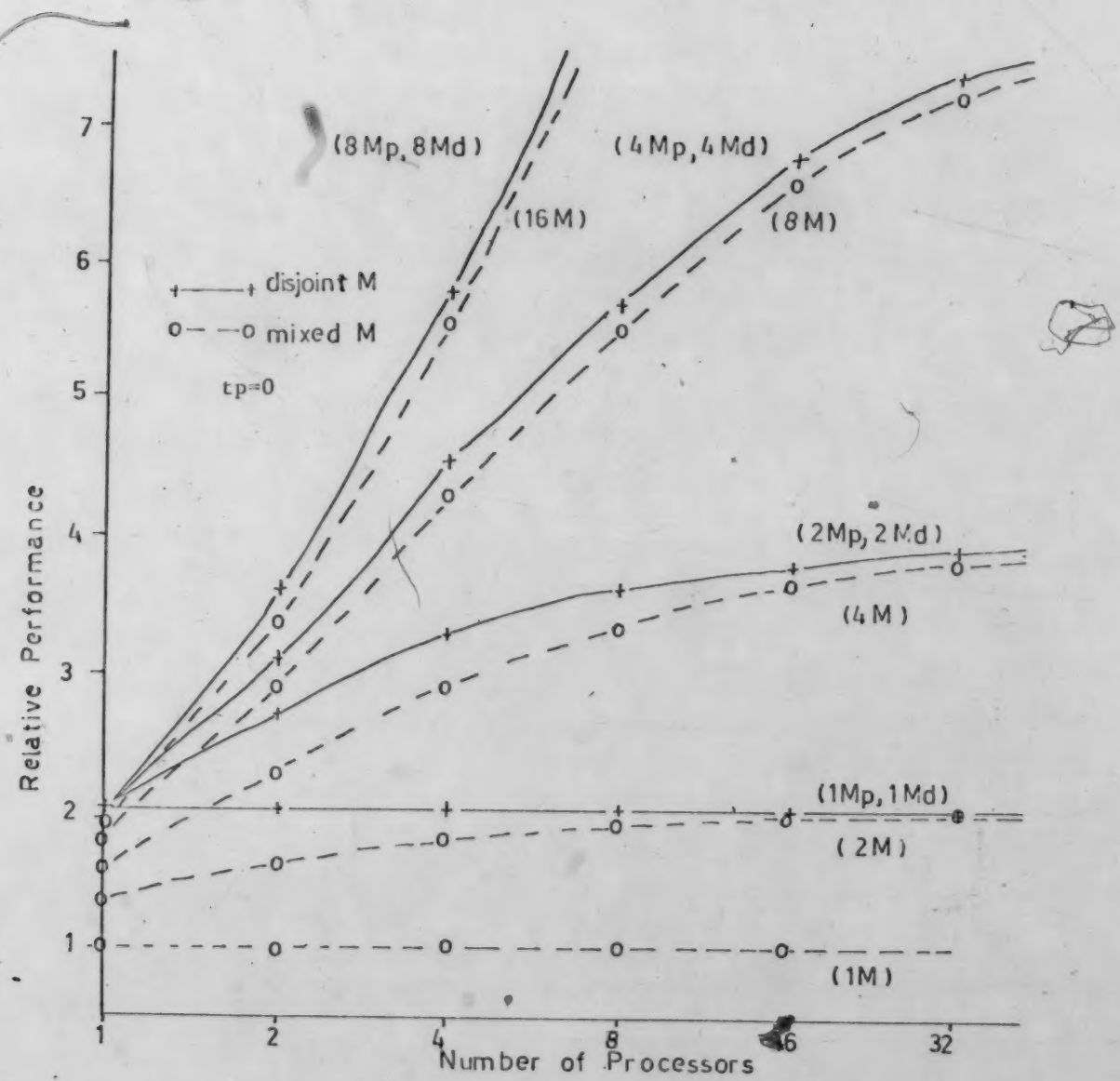


Figure 4.3a. Disjoint Memory vs. Mixed Memory Performance

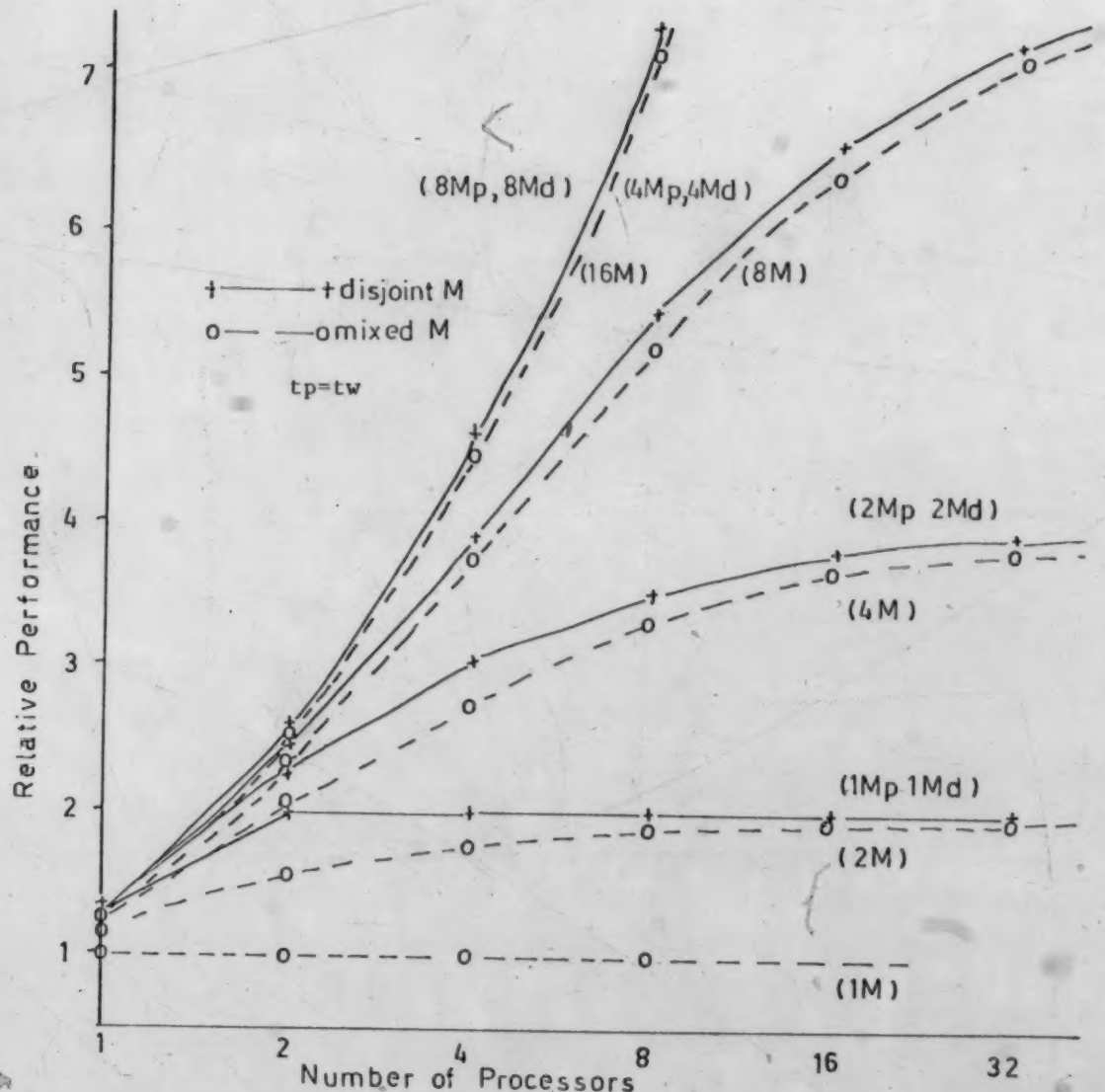


Figure 4.3b. Disjoint Memory vs Mixed Memory Performance .

how processor speed (tp) effects the designs. The results of both systems agree with intuitive thoughts about multiprocessors, ie. execution rate always improves, up to a limit, as memories or processors are added to the system. For a fixed number of memories, as processors are added to the configurations, both architectures approach the same limit in execution rate which is determined by the number of memories in the system and equals the total number of memories (m+k) times the execution rate of a uniprocessor with the same component speeds. With a large number of processors, the rate at which the memories can supply instructions and operands becomes the determining factor of the IER.

In all cases, the multiprocessor with disjoint memories out-performed the mixed memory system, the greatest difference occuring in configurations with small numbers of memories and in systems for which the ratio tp/tw is small, ie. fast processors and/or slow memories. The greatest improvement the disjoint memory gave over the mixed memory is about 54% (Figure 4.3a). For small memory systems, about 4 memories total, with 6 or less processors, a minimum improvement in execution rate of about 10% can be expected. For configurations larger than this, the difference between the two designs decreases from 10% while for smaller configurations, the difference increases. The reason the disjoint design out-performs the mixed design is that the disjoint design tends to lessen the memory access conflicts

producing a higher average memory occupancy. On the average, n/2 processors in the disjoint design will be fetching instructions from, say, k program memories and the other n/2 processors will be fetching operands from, again say, k data memories. This splitting of memory accesses reduces conflicts and produces a higher execution rate than the mixed memory design in which the n processors are all making fetches from 2*k mixed memories. This result can also be predicted from equation 3.12 which shows the average number of occupied memories.

The effect of disjoint memories on the execution rate of an SISD (one processor) computer can also be observed from Figures 4.3a,b. The disjoint memory always performs better than the mixed memory giving improvements ranging from near 100% for a very fast processor to as "little" as 25% for a slow processor system with tp=2tw.

An important role of simulation results as presented in Figures 4.3a,b and 4.4 is their use as a design tool. For example, Figure 4.4, which contains a subset of the data from Figures 4.3a,b plus other results, can be used to answer the question: Will doubling the speed of processors give a better performance increase than using twice as many slow processors? Assuming the number of memories remains fixed, going from two medium speed processors (tp=tw) to two fast ones (tp=.5tw) gives an increase in performance of about 12% (2.8/2.5). But using four of the medium speed processors, gives an increase

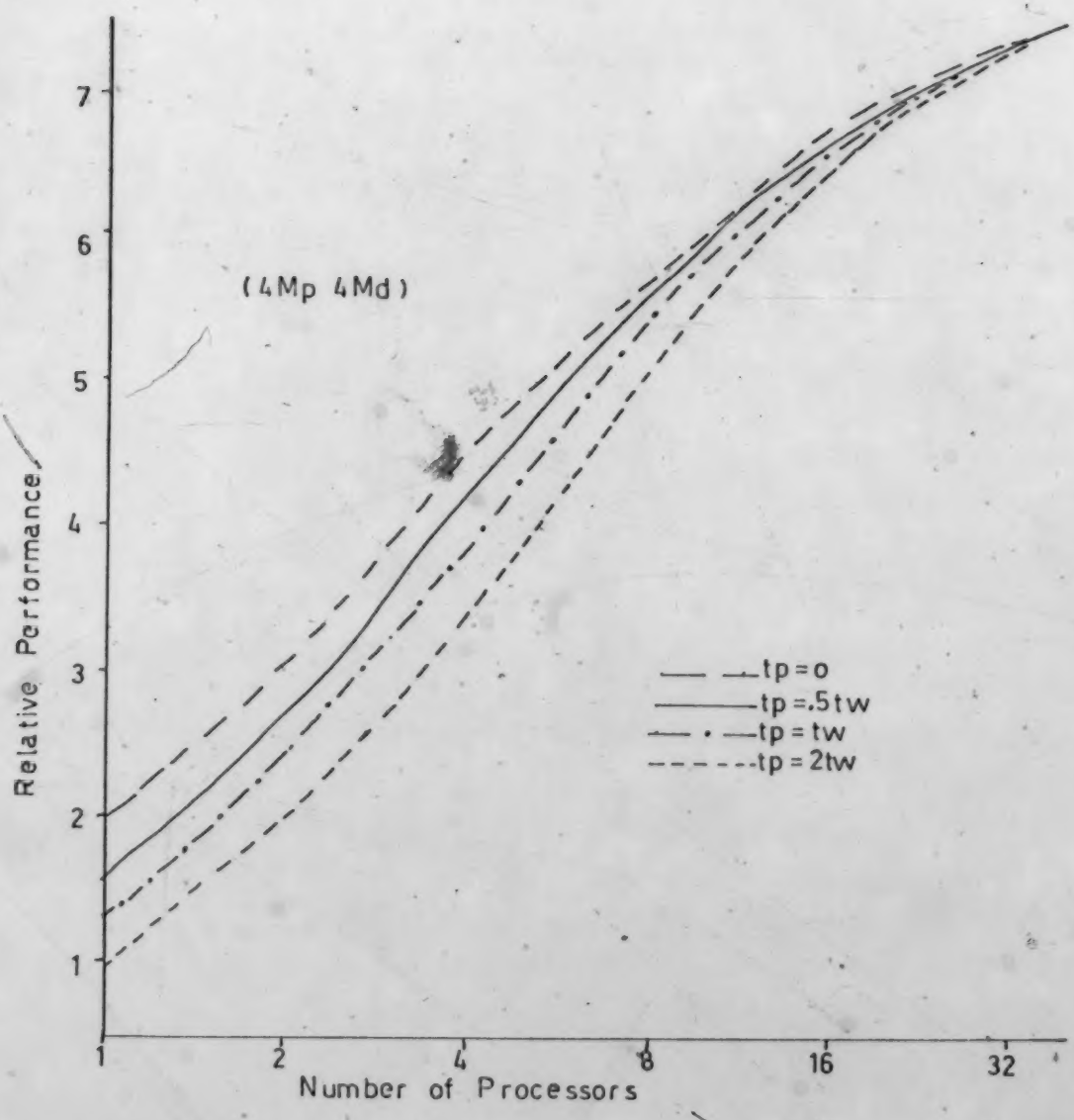


Figure 4.4. MIMD Performance with Disjoint Memories
(varying processor speed)

of about 60% (4/2.5) in the execution rate!

Simulation results for the MIMD computer with dedicated instruction memories are shown in Figures 4.5a,b for processor speeds tp=0 and 2tw, respectively. For comparison, the results of the multiprocessor with disjoint memories are also shown. For a very fast processor, Figure 4.5a indicates the dedicated memory provides a significant improvement of between 15% and 33% in execution rate over the disjoint memory for medium-sized "balanced" configurations, ie. four to eight operand memories and the number of processors and instruction memories equal to or less than the number of operand memories but greater than one. The reason the dedicated design produces higher instruction execution rates than the disjoint design is simply a reflection of the decrease in delays caused by decreased memory access conflicts as a result of removing the switch between the processors and instruction memories and dedicating an instruction memory to each processor.

These results are somewhat pessimistic toward the dedicated memory system. In the simulation the time required to switch from one memory to another is set to zero. In the randomly accessed instruction memory system, switching delays cause a decrease in the execution rate which is not reflected in Figures 4.5a,b. With dedicated instruction memories, no switching delays occur so that the results in Figures 4.5a,b reflect more closely the "real" dedicated instruction memory system.

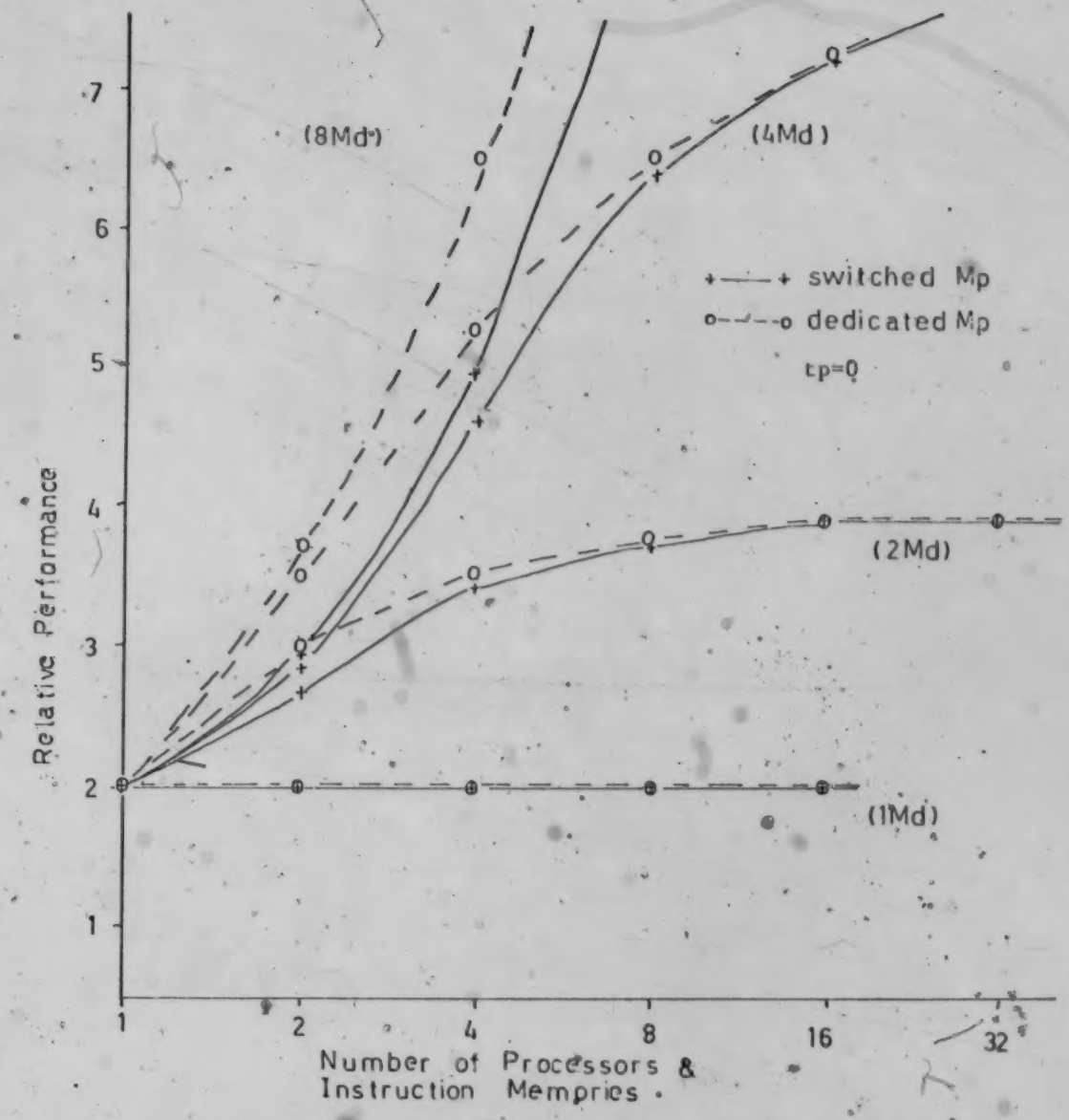


Figure 4.5a. Disjoint Memory vs Dedicated Program Memory Performance

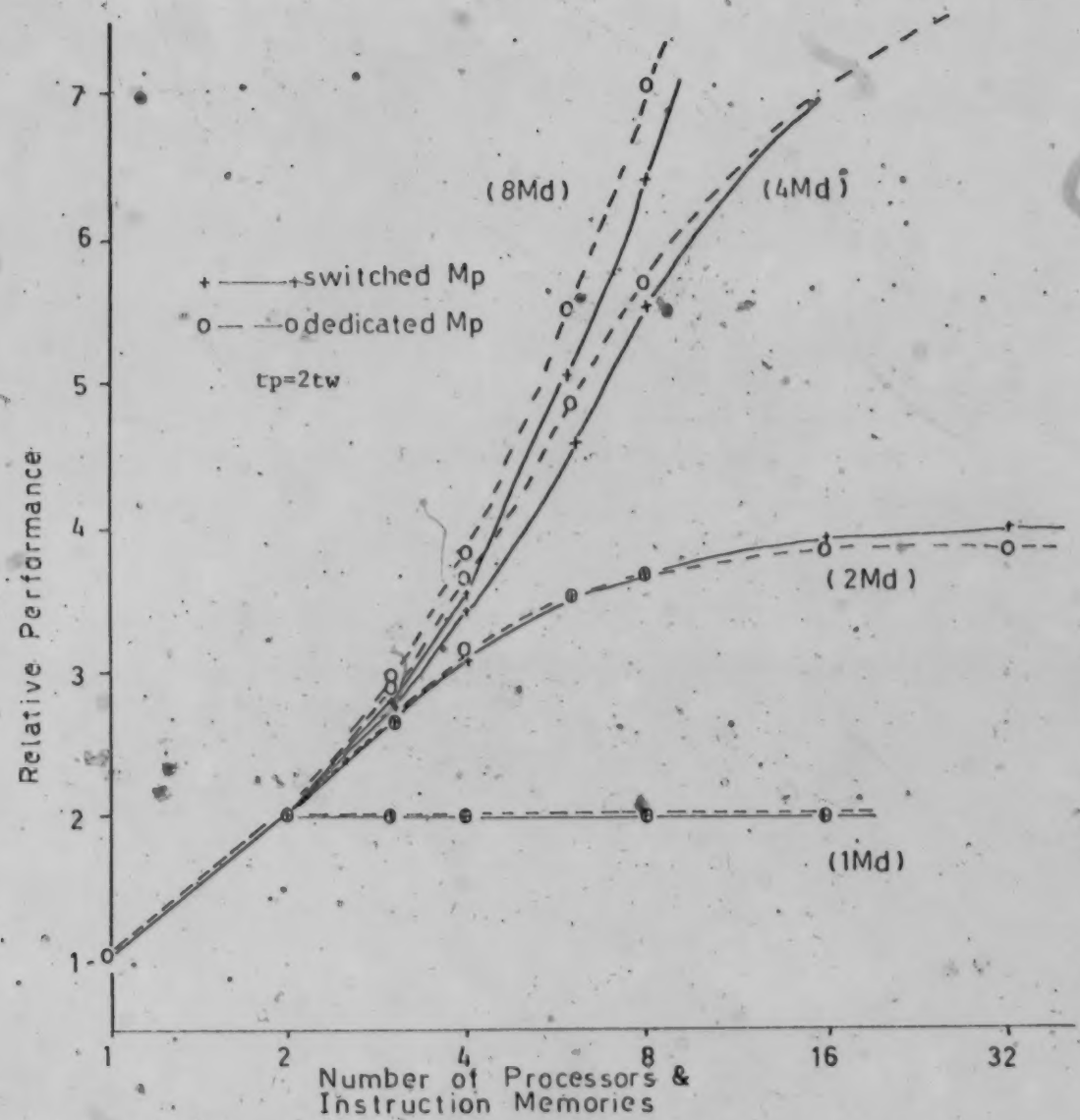


Figure 4.5b. Disjoint Memory vs Dedicated Program Memory Performance

As processor speed decreases the improvement in execution rate mentioned above rapidly diminishes so that for a slow processor, tp=2tw in Figure 4.5b, the maximum increase reported is only about 10%. For configurations with a small number of data memories, 2 to 4, and slow processors, little or no gain can be expected by changing from switched instruction memories to dedicated memories. This occurs because with the long processor delays, there is little or no queueing in either the data memories or the switched instruction memories. Thus processors experience little or no delay when accessing instructions from the switched system and the instruction memories appear dedicated.

An interesting phenomenon which did not occur in any other results obtained, appears in Figure 4.5b when only two operand memories are used. As a large number of processors and instruction memories are added to the system, the switched instruction memory system marginally out performs the dedicated instruction memory design. Compared to the other results, the systems appear to behave atypically in this instance. This is not so. The two operand memories are near their maximum fetch rate and many operand requests are queued. In either design, the delay through the operand memories is about the same. This is followed by another delay -- the time to execute the instruction -- which is also the same for both multiprocessors, as is the decode delay. The difference then must be attributed to the delay in the instruction memory

system. With dedicated memories, the minimum time between successive fetches is the memory cycle time, tc. With switched memories, a processor may successively access memories which are unoccupied giving a minimum time of ta, the memory access time, between fetches. Since ta<tc, the switched memory system can produce a higher execution rate. Although this is an interesting phenomenon, it should be noted that it occurred with only two operand memories and 8 or more processor/instruction memory pairs. Such a configuration is unlikely to be implemented in a real computer.

As was done for the switched program memory design, simulation data may be plotted as in Figure 4.6 and used as a design tool to study the effect of processor speed in dedicated memory multiprocessors. For the four operand memory configuration, Figure 4.6 shows that doubling the processor speed from tp=2tw to tp=tw gives a maximum improvement of about 30% for a one processor system. Doubling the speed again in a one processor system from tp=tw to tp=.5tw, gives another increase, but only about 23% this time. As processors and instruction memories are added, the gain made by increasing processor speed decreases so that as the operand memories reach their maximum fetch rate, around 8 processors with 4 data memories, increasing processor speed produces little or no improvement in the execution rate. The optimal execution rate in a multiprocessor with n dedicated instruction memories appears near the point where n/2 operand

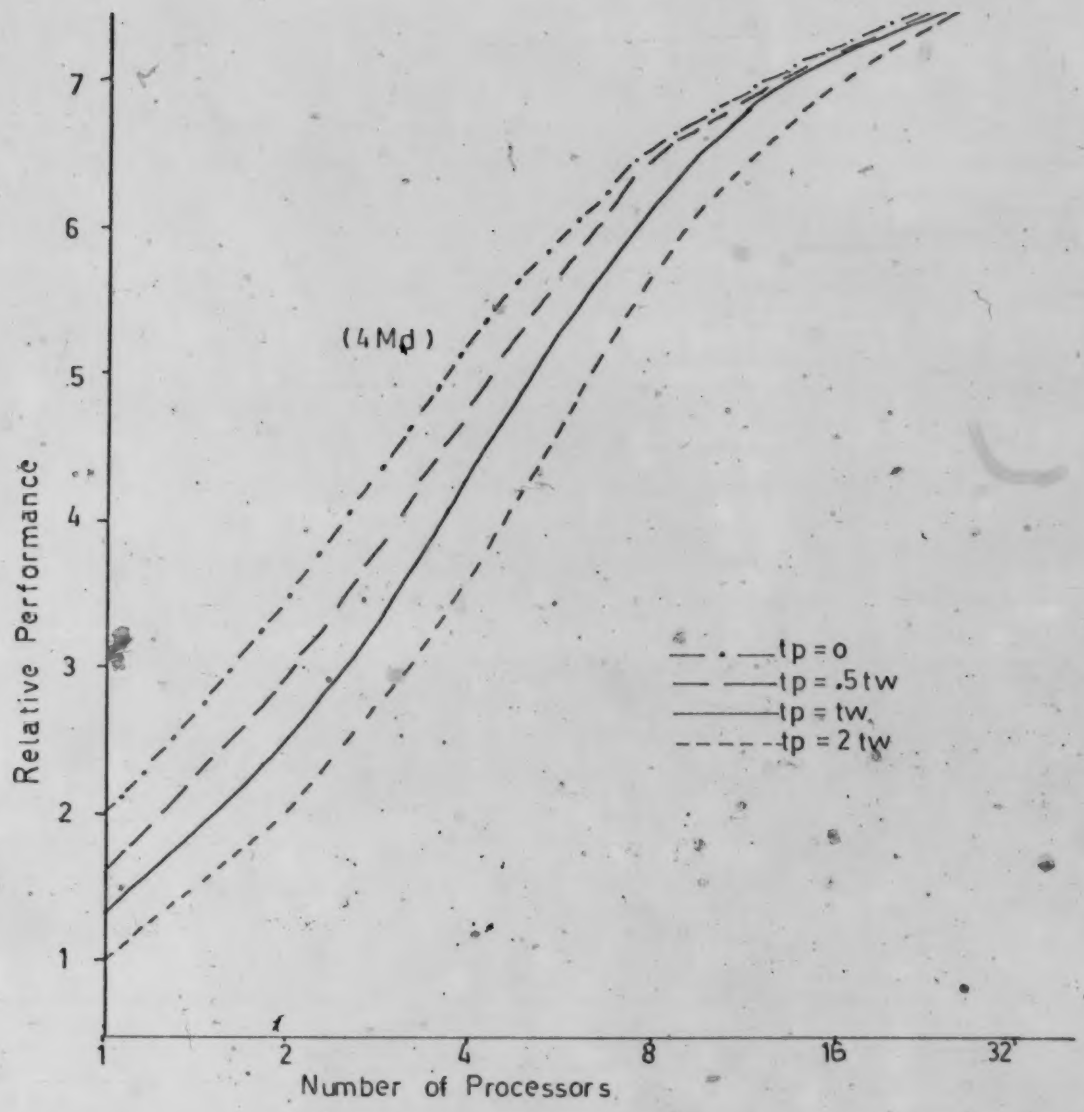


Figure 4.6. MIMD Performance with Dedicated Program Memories

memories are used.

As a design aid, the series of graphs presented in this section can aid in the selection of MIMD architectures required to meet certain criteria. For example, given four processors and twelve memory modules, Figures 4.3a and 4.5a show that a better execution rate is achieved by using four memories acting as dedicated instruction memories (with 8 operand memories) than by using six switched instruction memories (with 6 operand memories). The difference in execution rates is about 9% (5.2/4.8). (The value for the latter configuration being interpolated from Figure 4.3a.) The dedicated memory configuration produces a higher IER without requiring a switch between the processors and the program memories. This makes it even more attractive than the switched disjoint configuration. This of course assumes an instruction mix with equal numbers of operand fetches and instruction fetches. The object of the following section is to test this assumption, ie. the effect of instruction mix on relative execution rates.

4.3.3 Instruction Mix Results

The preceding sections have assumed that only one instruction with a single address format is executed in each multiprocessor design. In this section, the effect of this approximation in determining the relative execution rates of the different designs is tested by comparing simulation results obtained using a variety of instruction mixes.

As mentioned above, the simulator can handle four instruction formats with different relative frequencies of execution. Each instruction has an instruction fetch and decode phase. The action of the instruction after this phase determines its class. With reference to Figure 4.7, class 1 instructions are zero address instructions such as branches and register to register transfers. Class 2 contains instructions which require one operand fetch from memory, eg. load register, add from memory. Class 3 instructions are one word store type instructions. Instructions with two operand addresses, such as memory to memory moves, comprise class 4. Each of these may or may not have an execution time associated with it. In existing computers, a great variety of instruction types exist, proliferated by such concepts as index registers, base registers, stack architectures, etc. It is felt however that these basic four formats adequately represent the most commonly executed types of instructions in most present day computers.

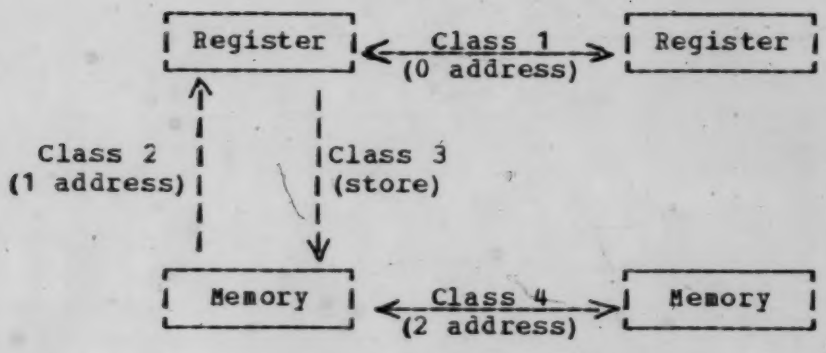


Figure 4.7. Instruction Classes

It is well known that instruction mix affects the execution rate in any computer. This thesis is mainly a comparative study of several computer designs and is thus concerned with "how well" one MIMD design executes a particular mix compared to another MIMD design. A secondary result, discussed further on, is the effect of instruction mix on each particular design.

To compare each design under the same workload (instruction mix), the execution rates were "normalized" with respect to the execution rate observed for that mix in a one processor, one memory (SISD) computer. Table 4.1, 4.2 and Figures 4.8(a-c) which show the results for some of the mixes tested, also indicate the mix executed. The times, tei, on the figures should be interpreted as nanoseconds. The simulation was run with memory cycle and restore times equal to 1000 and 500 ns, repectively. Each design tested contained

MIX NUMBER		1	2	3	4
class	tei	fi	fi	fi	fi
1	200	0.70	0.10	0.50	0.33
2	1400	0.10	0.10	0.00	0.33
3	200	0.10	0.10	0.50	0.33
4	200	0.10	0.70	0.00	0.00

Table 4.1. Instruction Mixes.

Components			MIX 1			MIX 2			MIX 3			MIX 4		
Pc	Mp	+ Md	MIXED	DISJ	DED	MIXED	DISJ	DED	MIXED	DISJ	DED	MIXED	DISJ	DED
1	1	1	1.21	1.23	1.22	1.29	1.49	1.49	1.30	1.50	1.50	1.19	1.34	1.32
2	2	2	2.19	2.06	2.41	2.31	2.42	2.50	2.27	2.31	2.85	2.21	2.35	2.52
4	4	2	4.08	3.81	4.80	4.38	4.38	4.55	4.28	4.05	5.50	4.25	4.31	4.86
8	8	8	7.95	7.19	9.48	8.50	8.35	8.70	8.28	7.77	10.8	8.31	8.30	9.55
16	16	16	16.7	14.2	18.9	16.9	16.3	17.1	16.5	16.0	21.3	16.4	16.0	19.0
			SISD:	0.68	MIPS	SISD:	0.37	MIPS	SISD:	0.67	MIPS	SISD:	0.51	MIPS

(+ For the mixed memory system, the number of memories is Mp+Md.)

Table 4.2. Relative Execution Rates for Instruction Mixes 1 to 4.

the same number of processors and data and instruction memories eg., 2Pc,2Md,2Mp. For the mixed memory system, the first row of numbers along the abscissa indicates the number of processors while the second row indicates the total number of memories in the configuration, eg. 2Pc,4M. From the previous results, these configurations could be regarded as "balanced" in number of components.

The instruction mixes simulated included heavy branching (70%), heavy memory-to-memory moves (70%), array initalizations, "typical" mixes and single address only instruction mixes.

With a high degree of branching, mix 1 in Table 4.2, the designs, ranked in order of high to low execution rates, are MIMD with dedicated program memories, mixed memories and disjoint memories. With this mix a high demand will be placed on the instruction memories. The dedicated memory system with no queueing delays can thus be expected to come out ahead. Although the disjoint memory system usually outperforms the mixed memory system, as indicated in previous sections, it does not in this case. In the mixed memory design, instruction requests are distributed throughout m memories while in the disjoint memory design, only m/2 memories are accessed for the same number of instructions. With the major portion of instructions generating no operand accesses, there will be more accessing conflicts and delays at the program memories in the disjoint memory system than in the mixed

memory system. This is one of two cases observed where the mixed memory outperformed the disjoint memory. In the other case, mix 3, the instruction mix was half class 1 (0 operand), half class 3 (store). There are more accesses to the operand memories in this mix than in the previous one but the processor execution time is small, putting a heavy demand on the program memories still. As stated above, these requests are distrubted over a larger number of memory modules in the mixed system than in the disjoint system resulting in shorter service delays.

Mix 2 consists primarily of memory to memory move type instructions. In the simulation, one instruction fetch of this type is followed by two accesses of the operand memories: one, a read, the other, a store. There is little processor execution delay in this mix but a great number of memory accesses. The results indicate little difference in the execution rates of the three design for this mix. All the memories are near their maximum fetch/restore rate so that the architectural differences of the designs make little difference in execution rate.

A comparison of the results for mixes 3 and 4 in Table 4.2 shows the effect of a not too radical change in instruction mix. Mix 4 includes one more instruction than mix 3 with a shift in execution frequencies. The dedicated memory system still produces the best execution rate but the disjoint and mixed memories results are reversed with the disjoint

memory system now performing slightly better than the mixed design. With the added instruction of class 2 (an instruction fetch followed by an operand fetch), the demand for memory cycles will shift somewhat from the instruction memories (in mix 3) toward the operand memories balancing the load. Under a more even load, the separated memories of the disjoint design cause fewer access conflicts resulting in a slightly better execution rate.

Mixes 5 and 6 shown in Figures 4.8a,b were chosen as more representative of an "average" mix. Mix 5 has one instruction of each class while Mix 6 has seven instructions representing all four classes also. The frequencies of mix 6 are based on a study[71] by R. O. Winder. Winder's paper reports frequencies for specific instructions executed in an RCA 70/45 computer. These instructions were classified into one of the four classes in Figure 4.7. Instructions of the same class and approximately equal processor execution time were then grouped to give the 7 instruction mix shown in Figure 4.8b. These mixes also show the characteristics mentioned above with the dedicated instruction memory producing the best execution rate followed by the disjoint memory system and then the mixed memory system.

Figure 4.8c shows the results for Mix 7, a single address format instruction mix. The results model those for other mixes, particularly the "typical" mixes, 5 and 6. The differences in the results of the mixes are mainly degree

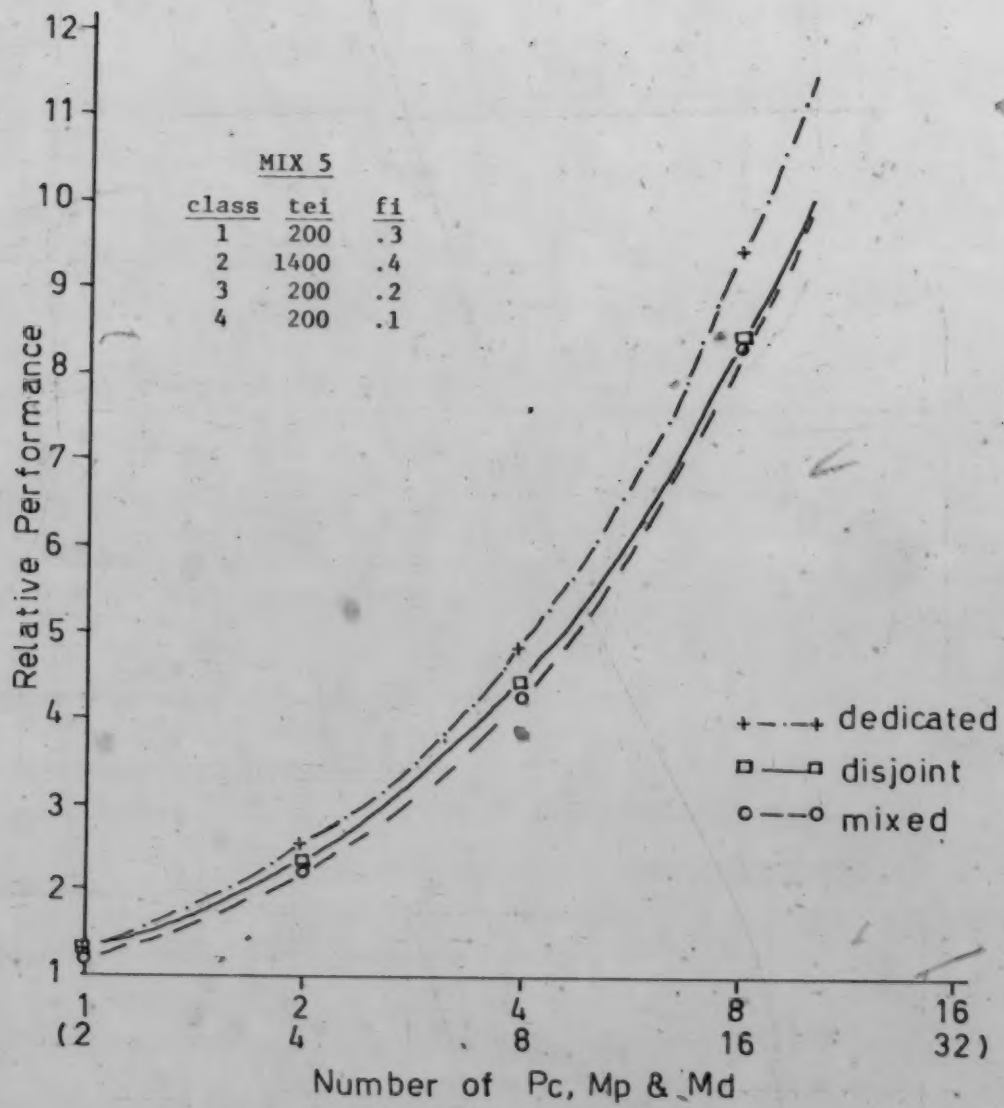


Figure 4.8a. Instruction Mix 5 Results

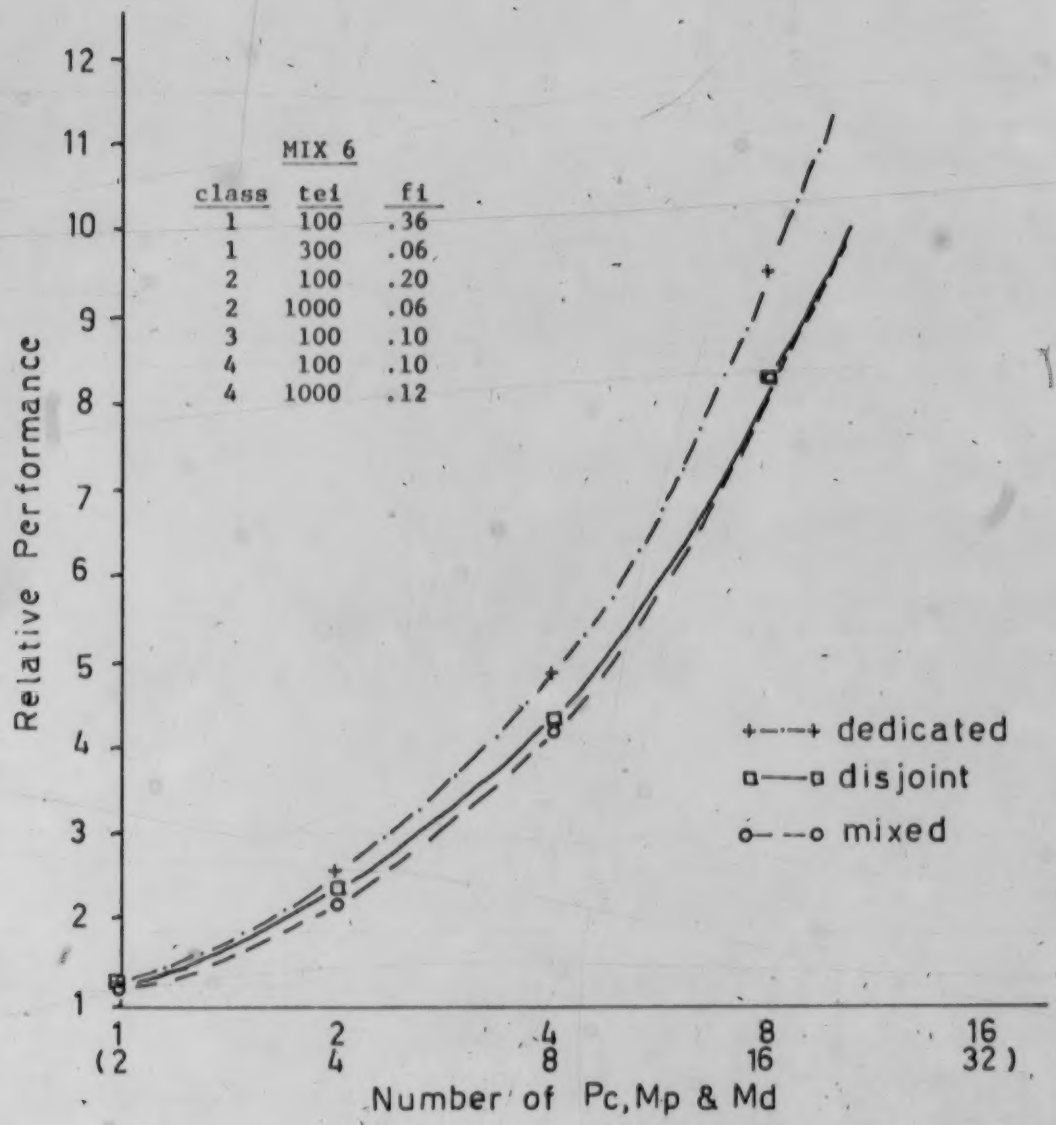


Figure 4.8b. Instruction Mix 6 Results

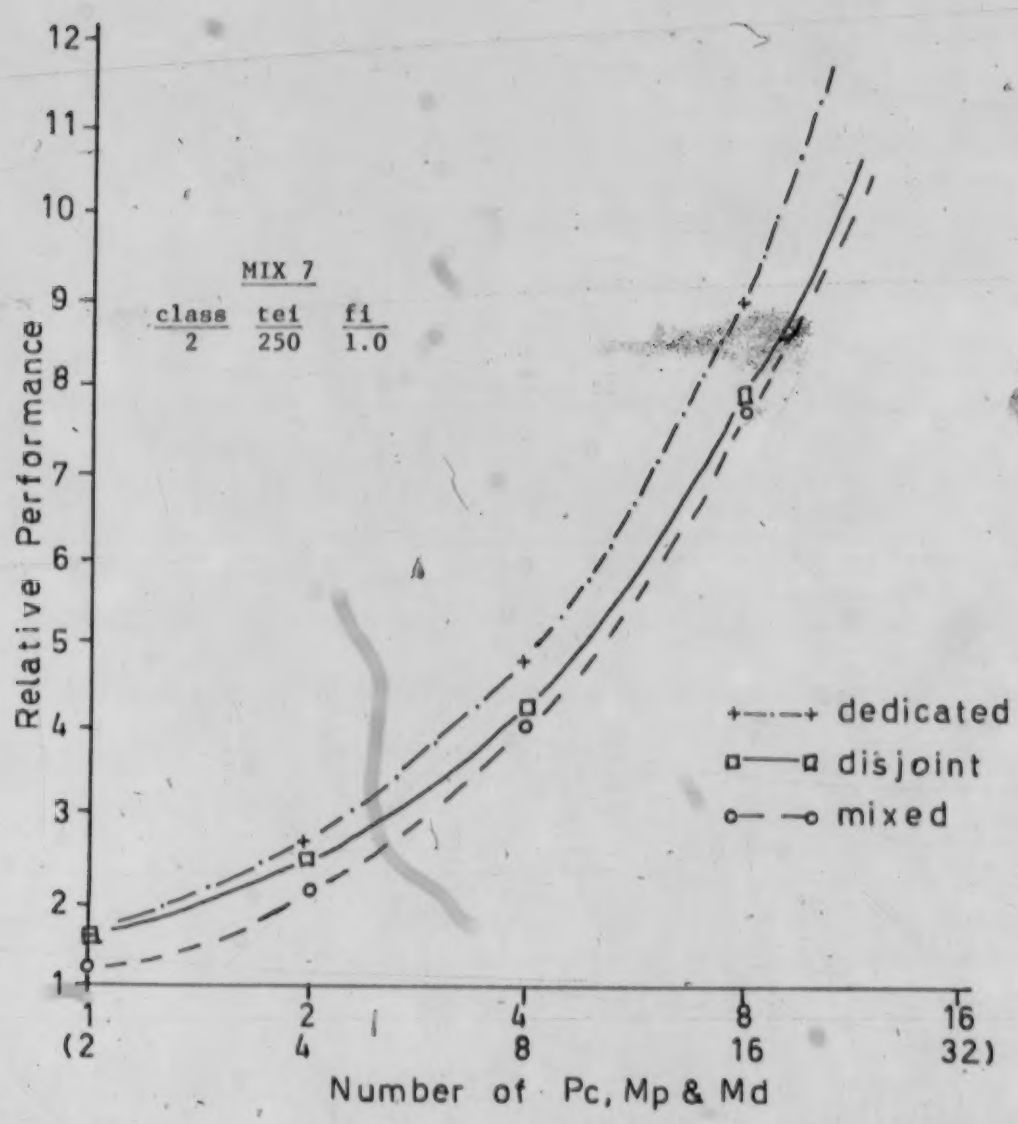


Figure 4.8c. Instruction Mix 7 Results

rather than character. The reason for this is that the mixes, in general, produce different instruction exection rates.

A final comment regarding Tables 4.2 and Figures 4.8(a-c) is that all the "curves" plotted are actually straight lines. It was stated above that these results are for balanced systems with n processors, n program memories and n data memories (2*n memories for mixed design). Since the proportions of memories and processors remain the same for different values of n, a change in n should produce a proportional change in the IER observed. This is in fact so. Each of the curves shown may be approximated by a linear equation IER=a*n+b with the different designs and mixes generating different slopes, a, and intercepts, b. As in previous results discussed, an important design consideration is the point where these execution rate lines intercept each other as it indicates the point at which one design starts to outperform the other even though both designs have equal number of components. The performance curve for the dedicated instruction memories usually intercepts the other curves at 1 processor and has a larger slope implying a better execution rate for larger configuations. The mixed and disjoint memory curves are more dependent on instruction mix and show a variety of intersection points. With mix 3, for example, the dedicated memory performs better than the mixed memory only for a one processor, one data memory, one program memory design. They produce the same execution rates for a 2

processor, 4 program memory, 4 data memory configuration and thereafter the mixed memory outperforms the dedicated memory. For "typical" mixes, Figures 4.8a and 4.8b, these two memory designs produce performance curves that intercept at large values of n (16 or more). This is in accord with similar observations made in Section 4.3.2.

To more clearly show the effects of instruction mix on each individual design, simulation results were plotted by memory design rather than by instruction mix, for a particular configuration. The results for each design are shown in Figures 4.9(a-c). Recall that mix 7 is the single address instruction mix and that the results for each mix are normalized with respect to the IER of a one processor, one memory computer executing that mix.

For an eight memory, mixed memory multiprocessor, Figure 4.9a shows relative execution rates ranging from 1.4 to 1.8 for only one processor. Interestingly, this range diminishes as processors are added until at about 4 or 5 processors a "balanced" system with eight memories, there is very little difference between the execution rates for the different mixes. At this point, the mixed memory system's relative execution rate is quite insensitive to instruction mix. This is not to say that different mixes produce the same execution rate. A wide range of IER's was observed for the mixes but relative to an SISD computer's performance, the mixed memory design's performance is independent of instruction mix. This

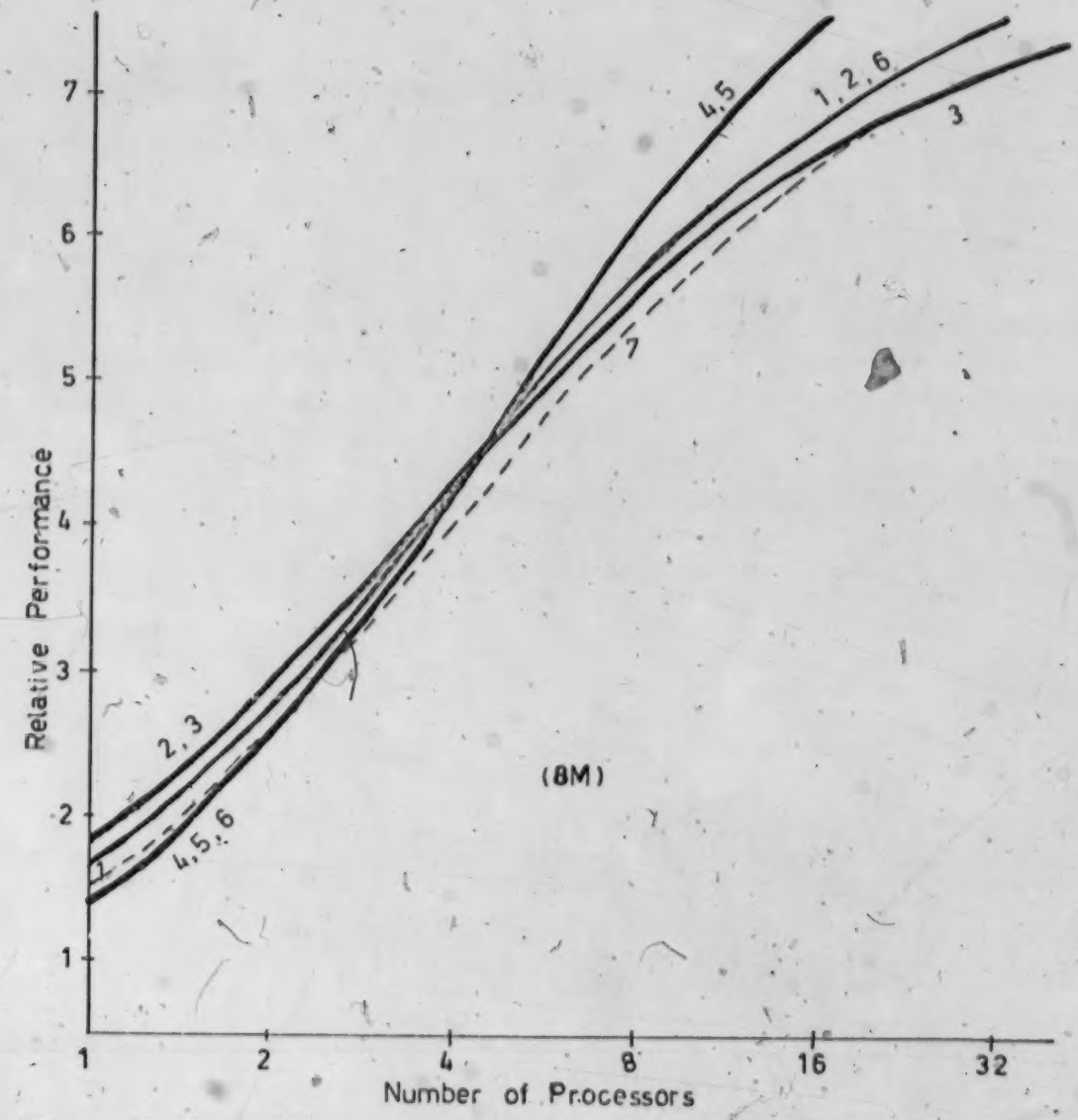


Figure 4.9a. Mixed Memory Performance for Different Mixes

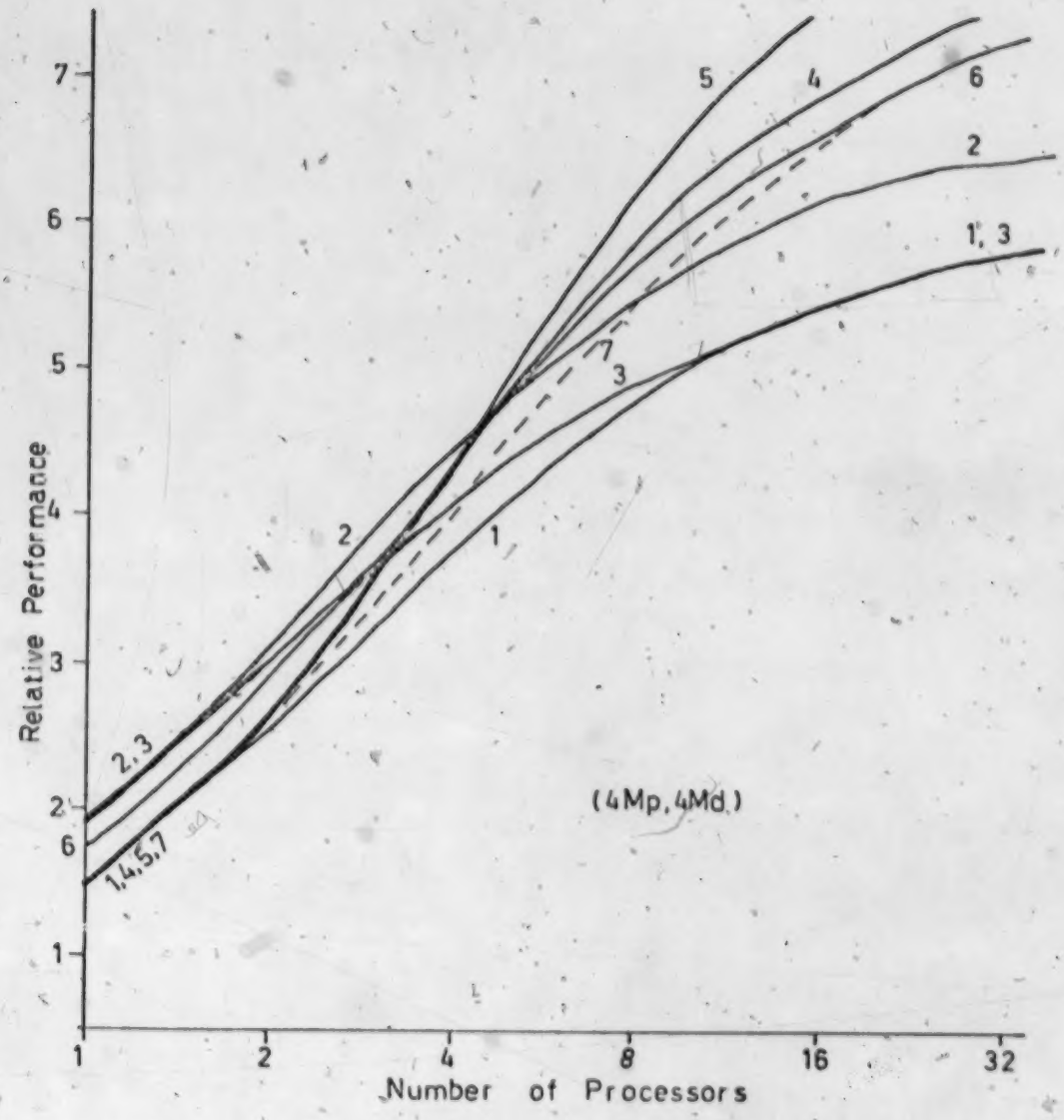


Figure 4.9b. Disjoint Memory Performance for Different Mixes

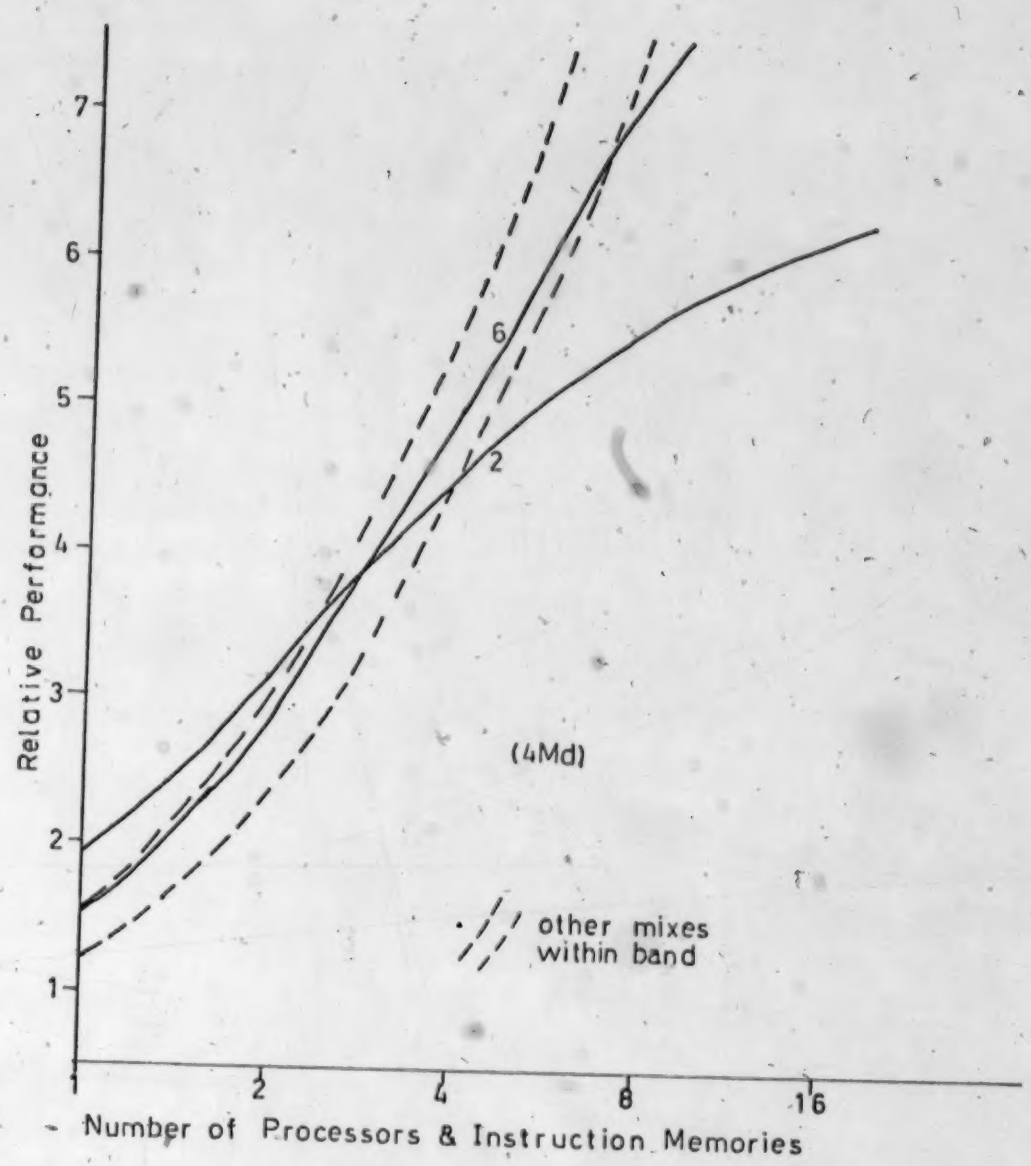


Figure 4.9c. Dedicated Program Memory Performance for Different Mixes

stability and predictability may be an important consideration when choosing an architecture which will be required to execute a wide variety of instruction mixes. As processors are added beyond this "balance" point, the instruction mixes again produce diverging performance curves. Mix 3 instructions have little processor execution delay, generating either instruction fetch requests or operand store requests. A heavy demand will be placed on the memories and this mix will be one of the first to "degrade" the performance as processors are added. Mixes 4 and 5, however, have good proportions of instructions which require long processor execution times. Thus the memories will have more time to complete their read/restore cycle before the processors can issue new read or write requests. A mix which causes a balance between memory speed and processor speed results in a better execution rate as shown in Figure 4.9a. Other mixes produce performance curves between these two extremes. Winder's mix (6) produces one of the lower IERs for a small number of processors but a "medium" IER for a large number of processors. The single address mix (7) has the same character of the other curves. A different value of tp for this mix may produce a more "average" curve.

Figure 4.9b shows instruction mix results for a disjoint memory multiprocessor with 4 program mixes and 4 data memories. The disjoint memory design produces a wider range of performance curves for the variety of instruction mixes

than the mixed memory system. As the configuration became more balanced in numbers of components, it was observed that the relative execution rate of a mixed memory multiprocessor was fairly insensitive to instruction mix. This is true of the disjoint memory design but to a lesser extent over a wide range of instruction mixes, the performance of the disjoint memory architecture relative to that of an SISD computer becomes quite unstable for configuration with more processors than program or data memories.

The simulation results for a multiprocessor with 4 dedicated instruction memories are shown in Figure 4.9c. With the exceptions of mixes 2 and 6, the execution rates were quite similar, lying within the band shown. Both mix 6 and, particularly, mix 2 make heavy demands on the operand memories. Thus as processors are added to the configuration more instructions are executed creating delays, with only small increases in the execution rate. This lack of improvement in IER is more prominent in the "unbalanced" mix 2 than in the more "balanced" mix 6. Again it is noted that the major divergence occurs when the system becomes overloaded with processors in relation to the number of operand memories. Mix 7, the single address mix again matched the results of most of the other mixes.

In summary, it has been observed that the relative execution rates of the three architectures are about the same regardless of the instruction mix if the configuration is

balanced in terms of the number of components. "Balanced" was discovered to mean a configuration with n processors, n instruction memories and n data memories or 2*n memories for a mixed memory system. The average processor execution time had little effect on relative execution rate for the variety of mixes tested. In the results obtained, the dedicated memory multiprocessor always performed the best of the three designs. The disjoint memory multiprocessor usually produced better execution rates than the mixed memory multiprocessor although the results here varied somewhat depending on instruction mix. Lastly, the choice of a single address format instruction for the analysis of the multiprocessor designs and their simulation using only this instruction format appears to be largely justified when compared to simulation results obtained using a variety of instructions and instruction mixes.

4.4 Utilization Results

Two series of runs were made to study resource utilization. The first series was aimed at balanced configurations, in particular, configurations with n processors, n data memories, and n program memories (2n memories for mixed design). The second series studies each of the three designs individually exploring utilization in unbalanced configurations. Although several instruction mixes were used to study utilization, the results of only one mix, mix 6, are used for illustration below. The conclusions

regarding this example however were found to apply to the other mixes as well.

For the balanced configurations studied in the first series of tests, the Instruction Execution Rate (IER) was found to follow a linear relationship. This is not the case for utilization. For increasing n, both processor and memory utilization decreases non-linearly toward a non-zero minimum. This decrease in utilization is caused by increased memory access conflicts.

Ranking the designs by IER or program memory utilization produced the same results, i.e., the dedicated instruction memory design had the highest Mp utilization and IER, the disjoint memory design had lower performance values and the mixed memory was lowest of the three. This is expected since each instruction execution required one instruction fetch. However ranking by data memory utilization does not necessarily produce the same result. Processor and data memory utilization are less stable with respect to instruction mix than program memory utilization since each instruction may or may not cause an operand fetch.

Table 4.3 and 4.4 show that for a balanced configuration with tp = .5tw, processor utilization is generally independent of the architecture. The range in processor utilization in Table 4.4 is mainly a result of instruction mix, not design. The processors are too fast for the memories in this case and

			MIXED			DISJOINT				DEDICATED			
Pc	Mp	Md	IER	uPc	uM	IER	uPc	uMp	uMd	IER	uPc	uMp	uMd
1	1	1	0.67	0.18	0.60	0.77	0.21	0.77	0.61	0.77	0.21	0.77	0.61
2	2	2	1.20	0.16	0.54	1.27	0.17	0.64	0.50	1.41	0.19	0.71	0.56
4	4	4	2.27	0.16	0.51	2.30	0.16	0.58	0.46	2.66	0.18	0.67	0.54
8	8	8	4.43	0.15	0.50	4.42	0.15	0.55	0.45	5.18	0.18	0.65	0.52
16	16	16	8.70	0.15	0.49	8.72	0.15	0.55	0.44	10.1	0.17	0.64	0.51

Table 4.3. IER and Utilization in Balanced Multiprocessor Configurations (Mix 6)

	IER		Pc Utilization			Mp Utilization			Md Utilization		
DESIGN	high	low	high	low	diff*	high	low	diff*	high	low	diff*
MIXED	10.73	0.48	0.26	0.12	0.04	0.62	0.47	0.14	--	--	--
DISJOINT	9.61	0.56	0.27	0.12	0.08	0.83	0.38	0.32	0.88	0.24	0.32
DEDICATED	12.77	0.56	0.27	0.13	0.06	0.83	0.40	0.24	0.89	0.32	0.25
SISD	0.68	0.37	0.22	0.12	--	1.00	0.94	--	--	--	--

(* largest difference within any mix tested)

Table 4.4. IER and Utilization Ranges for Various Instruction Mixes

are in general under-utilized. This under-utilization accounts for the similarity in processor utilizations. Both memory utilizations have similar high and low values in Table 4.4. This is not to imply that the memory designs have similar utilizations as these maxima and minima are produced by different mixes.

A more important comparison is the range between the high and low utilizations. For both instruction and operand memories, the dedicated program memory design produced a smaller range in utilization than the disjoint memories. It is the more stable design with respect to utilization over a wide range of configurations and instruction mixes. The use of dedicated program memories over disjoint memories has a direct effect in creating a higher utilization of operand memories. The reason is that since the dedicated Mp design has a higher execution rate more accesses are made to its operand memories in a period of time producing a higher utilization. This is observed in Table 4.3 also.

The mixed memory design produces an "average" utilization figure since it is based on both instruction and operand fetches. In general, it was observed, as in Table 4.3, that the average memory utilization for the dedicated or disjoint memory designs was higher than that of the mixed memory design. This is a result of the decrease in memory access conflicts and the higher IER in the disjoint and dedicated program memory designs.

In a second series of tests the characteristics of utilization were observed for each individual design when the number of components in the configuration were varied. The results of some of these tests are shown in Figures 4.10 (a-c). In Figure 4.10a, utilization was observed for a mixed memory design with 8 memory modules. Because of the similarity of the IER and memory utilization (uM) curves, the ratio IER/uM was calculated at a number of points. For each mix tested, IER/uM observed to be constant. With any mix, the relative difference between the maximum and minimum values of IER/uM was observed to be a maximum of 2%. This could be attributed to small variations in the simulated results. Over a series of four mixes, IER/uM was observed to be in the range (3.89, 5.36). This ratio appears to be very closely related to the number of memories in the system: IER/uM = (number of memories)/2. In fact the single address mix, the IER/uM ratio is in a configuration with 8 memories 4. This close agreement can be attributed to the fact that there is one operand fetch per instruction fetch in the single address mix. Although this relation may not be too important in this study, its existence may be used to study the instruction mix in a particular configuration for which execution rate and memory utilization are known.

In the disjoint memory design, Figure 4.10b, the IER and data memory utilization (uMd) curves are strikingly similar. The closeness is simply a coincidence of scales but the

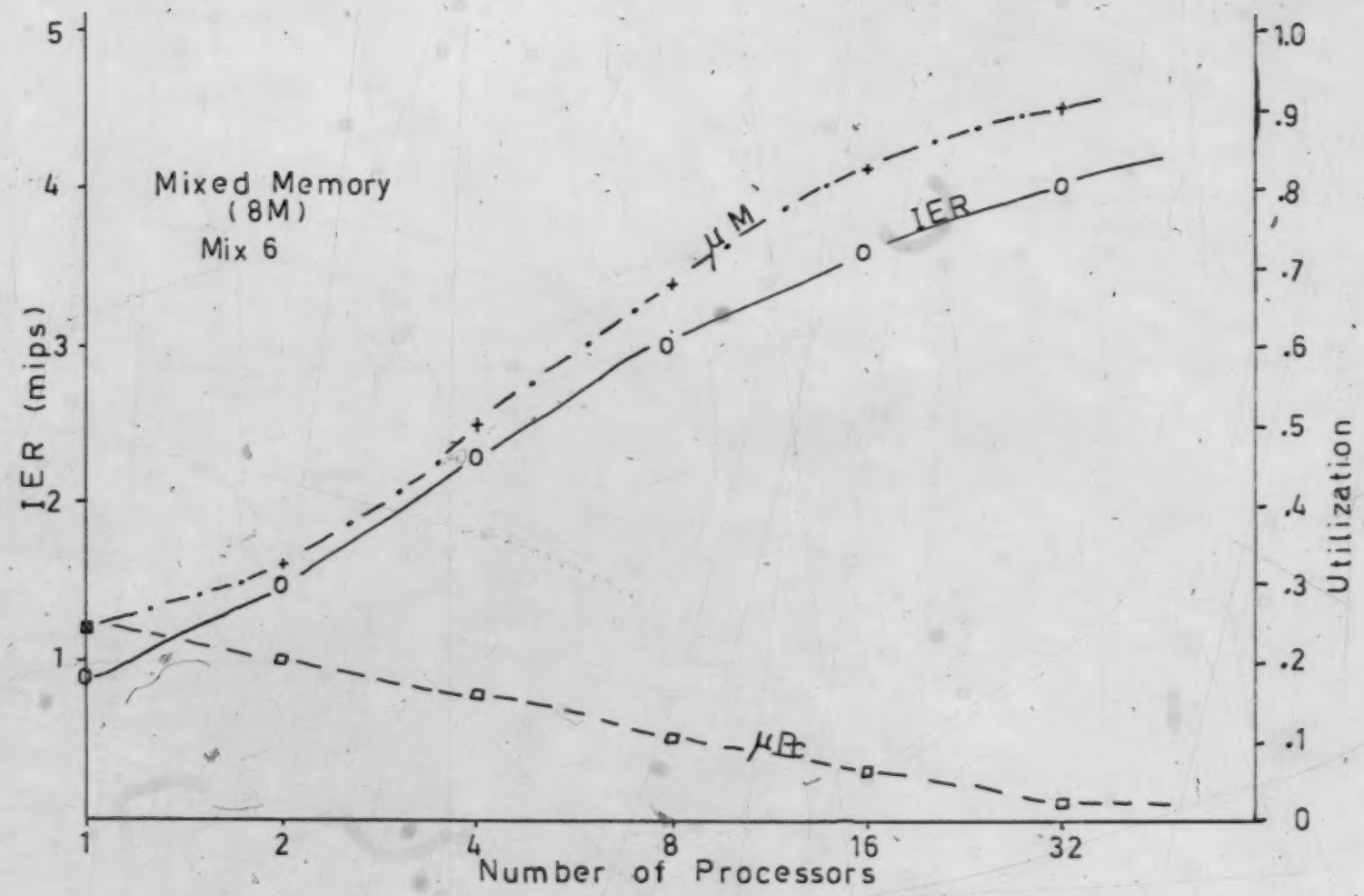


Figure 4.10a. IER and Utilization for Mixed Memory Multiprocessor

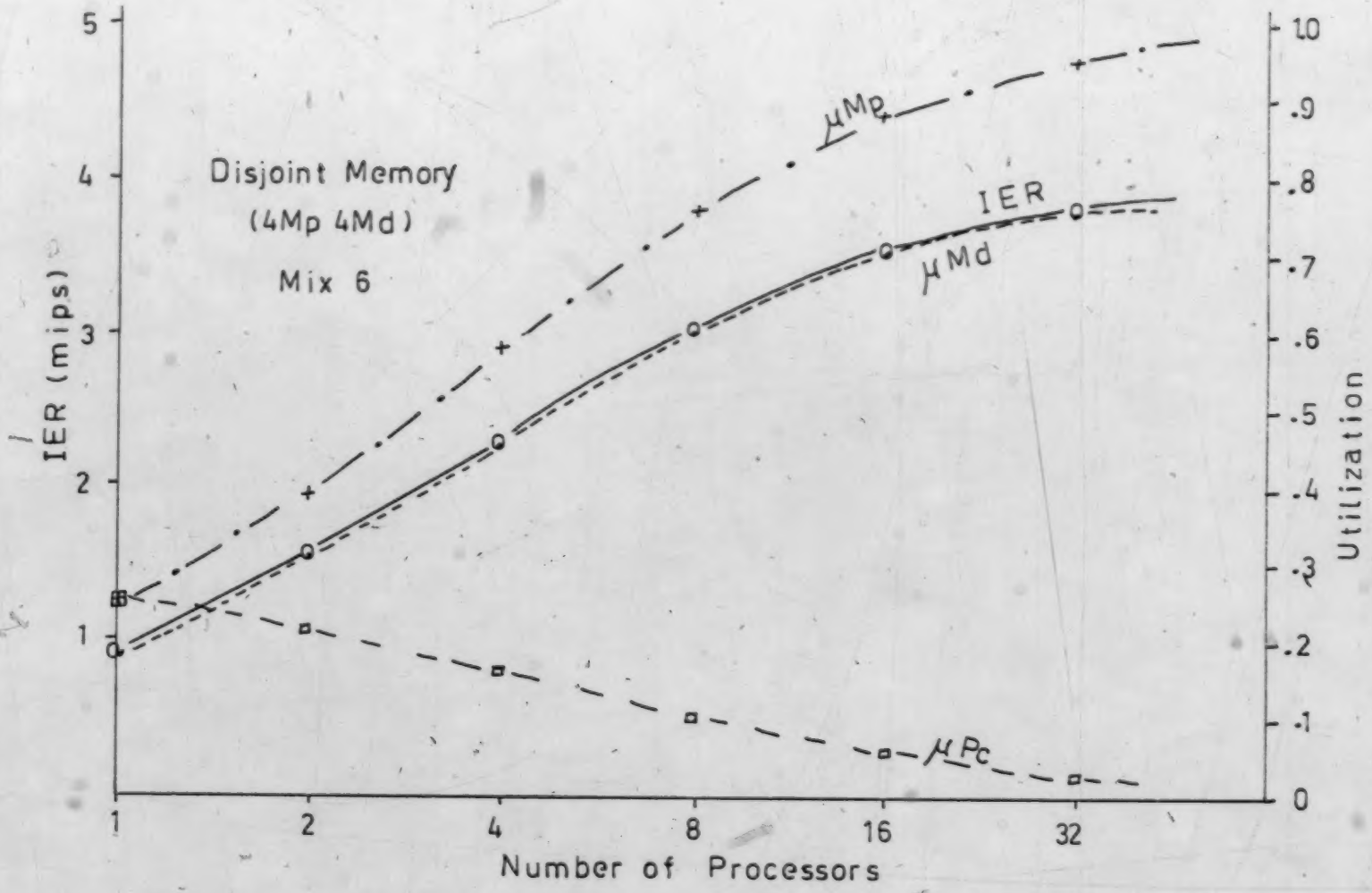


Figure 4.10b. IER and Utilization for Disjoint Memory Multiprocessor

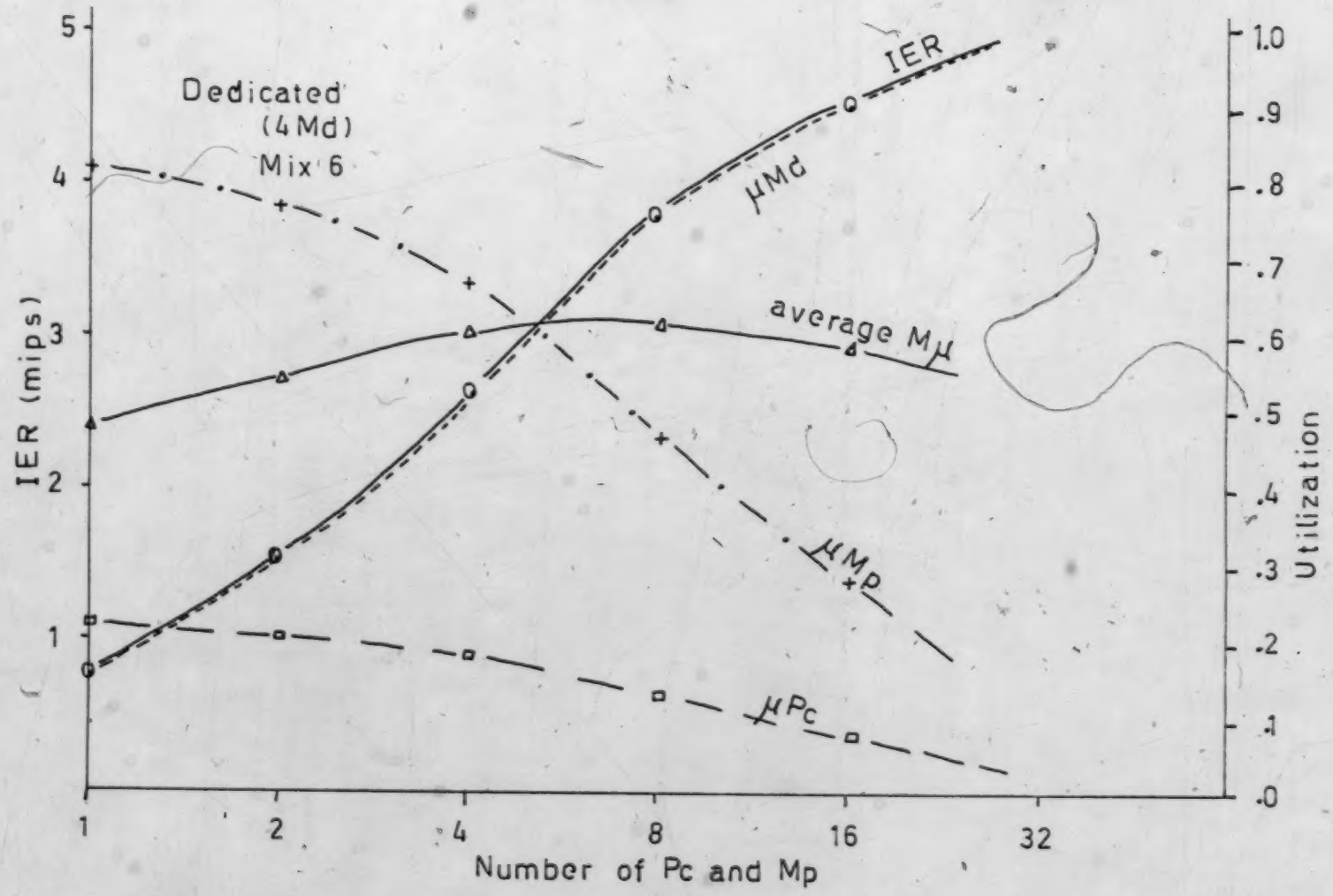


Figure 4.10c. IER and Utilization for Dedicated Program Memory Multiprocessor

character of the two curves is not, for it was discovered as for the mixed memory design that the ratio IER/uMd is approximately constant for each instruction mix. The value of IER/uMd was in the range (3.96,8.14) for the mixes tested, the minimum resulting from the single address mix (mix 7) while the maximum was observed for a 50% zero address and 50% store instruction mix (mix 3). For the program memory system however almost no variance in the IER/uMp ratio (3.93,4.02) was observed over the variety of mixes or configurations. Since each instruction requires one instruction fetch , this ratio might be expected to be independent of mix. It is observed to equal the number of instruction memories in the configuration, 4 in this case.

For the dedicated instruction memory system with 4 data memories (Figure 4.10c) the IER/uMd ratio was observed to behave as it did for the disjoint design, the IER/uMd ratios were almost the same even though the dedicated system had higher IER and utilization. Thus this ratio is purely an indication of the character of the instruction mix. Similar to the disjoint memory design, the IER/uMp ratio indicated the number of processor/instruction memory pairs in the configuration. Again this ratio was observed to be independent of mix. In the dedicated instruction memory system, uMp is seen to fall off rapidly as processor/memory pairs are added to the system. This is the result of having to dedicate program memories to each processor in this design.

As processor/program memory pairs are added to the configuration the data memories become the bottleneck to the instruction processing causing both uPc and uMp to decrease. Although the dedicated program design produces the best execution rate compared to the other design studied, this poor utilization characteristic may eliminate its choice for certain applications.

4.5 Summary

In this chapter, the results of several simulation studies are reported. One study comparing the analytic results of Chapter 3 to the simulation results, found varying degrees of agreement between the various models. The simulation and analytic models of the dedicated instruction multiprocessor had good agreement while the models of the generalized disjoint memory multiprocessor were not as close to each other. The reason for this was attributed to the compounding of approximations for more complex disjoint design. The three multiprocessor designs were compared in some detail under the assumption that each was executing a single address instruction mix. It was found that the dedicated instruction memory design produced the best rate of execution of the three designs over a wide range of configurations. The disjoint memory system was next with a somewhat lower IER but usually above the execution rate achieved by a multiprocessor with a mixed memory system. In

order to substantiate the results based on a single address instruction mix, the designs were also tested using instruction mixes composed of four types of instructions. Mixes using up to seven instructions with various processor execution times were tested. The results tend to agree with the general conclusions obtained using the single address format. Some mixes were discovered however which produce poor performance results. Such mixes, especially for unbalanced configurations, tend to occur infrequently in a real computer [71].

Component utilization in the three multiprocessor designs was also studied to a lesser extent. It was found that in "balanced" systems the component utilization decreased somewhat for larger configurations even though the IER increases. The reason for this is increased memory access conflicts as the system grows. It was also shown that IER and memory utilzation were directly proportional to each other. Because each instruction execution requires one instruction fetch a direct relation between number of instruction memories, IER and instruction memory utilization was observed. This was not so with respect to the data memory system. Although a linear relationship between IER and data memory utilizatien was observed, it depended on instruction mix. Conversely, given IER and uMd, one may be able to infer characteristics of the instruction mix.

In the next chapter, conclusions which may be drawn from

the results in this chapter are summarized. Suggestions for further research in this field are also proposed.

CHAPTER 5

CONCLUSION

5.1 Summary and Conclusions

As outlined in Chapter 1, the objectives of this thesis are twofold. One is the development of multiprocessing models which can be used as design tools. The other objective is the study of memory architecture effects on instruction execution rate and utilization. Three multiprocessor designs having mixed, disjoint and dedicated program memories were studied. In Chapter 3, analytical models were developed and their results compared to simulation results in Chapter 4. Each design was simulated using a wide variety of instruction mixes in order to test the effects of instruction mix on execution rate and utilization and to determine the effect of assumptions made in develping the analytic models.

It was illustrated in Chapter 4 that the models developed can provide valuable insight into multiprocessor behaviour when used as a design tool. Examples were given by posing and answering questions such as "Will doubling the number of processors produce the same change in IER as doubling their speed?". In another case it was observed that equal increments in processor speed do not necessarily produce proportional increments in the IER. In comparing the result for the three memory architectures under a variety of

instruction mixes, the dedicated program memory design was found to be more independent of mix for both instruction execution rate and utilization. The mixed memory design was also found to produce similar relative execution rates for the variety of instruction mixes. This is especially true when the number of different components are "balanced", i.e. an n processor, n program memory and n data memory configuration. The disjoint memory design is more sensitive to instruction mix with respect to IER even for balanced configurations. Even though the disjoint memory design usually produces a higher IER than the mixed memory design, its instability may make it an unsuitable architecture for environments with large differences in instruction mix. The dedicated program memory design produced the highest execution rates over the mixes tested and was observed to produce an IER largely independent of mix. Some mixes did however produce poor execution rates although they did so in the other designs as well.

An important design point discovered for balanced multiprocessor configurations is that the IER is a linear function of the number of components, i.e. IER=a*n+b where a and b are parameters that are functions of memory design and instruction mix. This knowledge allows easy evaluation of the IER for any number of components once two points are established through simulation or analysis. For different memory architectures, these performance lines have different slopes. The points at which these lines intersect for the

different designs are important in multiprocessor design as they indicate points of "trade off" where systems with equal numbers of components start to produce different performances. When resource utilization was studied it was discovered that the ratio IER/uMd (data memories utilization) was a constant for each mix tested in either the disjoint or dedicated memory design. That is, the ratio IER/uMd is purely an indication of instruction mix character. Although not pursued in this study, this relation may be useful as a method to infer instruction mix characteristics in an operating environment.

Although the above comments are based on simulation results, analytical models play a very important role in computer design in two ways. One, they can provide some insight into computer operation. Second, from a purely computational point of view they can produce results more economically than simulations. In comparing the analytical models to simulations, close agreement was found for the dedicated memory design. Lesser agreement was observed between the disjoint memory models. This was attributed to the "compounding of errors". In the dedicated memory architecture accessing conflicts occur at only the data memory set but in the disjoint memory design, accessing conflicts occur at both sets, the program and data memories. All the models agreed in their asymptotic performances as large numbers of processors were added since the memory system is forced to perform at its limit. The maximum relative

performance equals the number of memory modules.

In comparing the three multiprocessor designs to each other, it was found that for most instruction mixes, the dedicated Mp design produced the highest IER, followed in order by the disjoint memory and the mixed memory designs. Exceptions to this ordering were caused by mixes with uneven distributions of instructions. A mix which generates a large proportion of instruction accesses for example was seen to make the mixed memory design outperform the disjoint memory design. For "typical" mixes and the single address mix, the rankings were as noted originally. The disjoint design produced execution rates up to 50% greater than the mixed memory design for small numbers of memories or for which the ratio tp/tw is small. The dedicated Mp design offered yet another increase in IER; between 15 and 33% greater than the disjoint memory performance for "balanced" configurations. Thus it appears that the disjoint and dedicated program memory multiprocessor designs offer significant improvements in processing power over the mixed memory design.

5.2 Suggestions for Further Research

The results of the analytic and simulation models for the disjoint memory design did not agree closely. Which model more closely reflects the "true" behaviour of the design can only be tested by studying a real multiprocessor. Presently there are very few real multiprocessors in existence and few

results are published making it diffult to compare the models in this thesis with real machines. One recent paper [53] reports both observed and simulated performance of an existing multiprocessor with an architecture very similar to the disjoint design proposed here. Unfortunately, the paper is not complete and some parameters which affect the IER are not shown. Several runs of the simulator used in this thesis were made using, where possible, parameter values stated in [53] and assumed values where none were given. The results agreed favorably with those reported in [53] but because of the assumptions made, they are not reported in detail here. A more thorough comparison is suggested. Alternatively, a more detailed analytic model or a model based on a different approach may be developed and its results compared to existing results. Such research is open-ended.

Another avenue of research is a performance/cost study of the proposed designs. A good case for such a study is shown in the utilization results of Chapter 4. In balanced configurations, component utilization was observed to decrease while the IER increased as components were added. This behaviour may produce conflicts in design criteria, eg. high IER and high utilization. In such a case the IER/cost ratio may be a better performance measure.

The inference of instruction mix characteristics as reflected by the IER/uMd ratio is a third research topic. Monitoring instruction execution in a working computer is a

somewhat difficult and expensive process [71]. If one is able to measure the IER and operand memory utilizations dynamically, then the IER/uMd ratio may provide insights into how the computer system is being used.

Lastly, the extension of the simulation models to provide more detailed information of multiprocessor operation is suggested. Such studies could include, for example, parameter for I/O activity and the effect of a switch with concurrency of less than min(m,n). Extensions could also be made to the types of instructions and their methods of execution.

REFERENCES

1. Batcher,K.E., "STARAN Parallel Processor System Hardware", Proc. AFIPS 1974 NCC, pp.405-410.

2. Baer,J.-L., "A Survey of Some Theoretical Aspects of Multiprocessing", Computing Surveys, Vol.5,No.1,(March 1973), pp.31-80.

3. Baer,J-L., "Large Scale Systems", in Computer Science, A.F.Cardenas, L.Presser, and M.A.Marin (eds.), John Wiley and Sons, 1972.

4. Baer,J-L. and Estrin,G., "Bounds for Maximum Parallelism in a Bilogic Graph Model of Computations", IEEE Trans. Computers, Vol.C-18, No.11, (November,1969), pp.1012-1014.

5. Barnes,G.H., et al., "The ILLIAC IV Computer", IEEE Trans. on Computers, Vol.C-17, No.8, (August,1968), pp.746-757.

6. Barr,W.J., "Cost Effective Analysis of Network Computers", University of Illinois at Urbana-Champagne, Report UIUCDCS-R-720538, (August,1972), pp.20-25.

7. Bell,C.G., Chen,R. and Rege,S., "Effect of Technology on Near Term Computer Structures", Computer, Vol.5, No.2, (March/April,1972), pp.29-38.

8. Bell,C.G. and Newell,A., Computer Structures: Readings and Examples, McGraw-Hill, 1971.

9. Bell,C.G. and Newell,A., "Possibilities for Computer Structures 1971", Proc. AFIPS 1971 FJCC, Vol.39, pp.387-394.

10. Bell,C.G. et al., "C.mmp: The CMU Multiminiprocessor Computer", Carnegie-Mellon University Report CMU-CS-72-112, August, 1971.

11. Bloch,E., "The Engineering Design of the Stretch Computer", Proc. EJCC, 1959, pp.48-59.

12. Bonseigneur,P., "Description of the 7600 Computer System", Computer Group News, Vol.2, No.9, (May,1969), pp.11,15.

13. Borck,W.C., McReynolds,R.C. and Slotnick,D.L., "The SOLOMON Computer", Proc. AFIPS 1962 FJCC, Vol.22, pp.97-107.

14. Bouknight,W.J. et al., "The ILLIAC IV System", Proc. IEEE, Vol.60, No.4, (April,1972), pp.369-388.

15. Burks,A.W., Goldstine,H.H. and von Neumann,J., "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument", Part I, Vol.1, Rept. prepared for U.S. Army Ordnance Dept., 1946, in Collected Works of John von Neumann, A.H.Taub (ed.), Vol.5, pp.34-79, Pergamon Press, 1961.

16. Burnett,G.J., Koczela,L.J. and Hokum,R.A., "A Distributed Processing System for General Purpose Computing", Proc. AFIPS 1967 FJCC, Vol.31, pp.757-768.

17. Chamberlin,D.D., "Parallel Implementation of a Single-Assignment Language", PhD. Thesis, Stanford University, 1971.

18. Chen,T.C., "Parallelism, Pipelining, and Computer Efficiency", Computer Design, Vol.10, (1971), pp.69-74.

19. Chen,T.C., "Unconventional Superspeed Computer Systems", Proc. AFIPS 1971 SJCC, Vol.38, pp.365-371.

20. Covo,A.A., "Analysis of Multiprocessor Control Organizations with Partial Program Memory Replication", IEEE Trans. on Computers, Vol.C-23,No.2,(February 1974),pp.113-120.

21. Ellis,C.A., "Parallel Compiling Techniques", Proc. ACM 1971 Annual Conf., ACM, New York, pp.508-519.

22. Farber,D.J. and Larson,K.C., "The System Architecture of the Distributed Computer System -- The Communications System ", Symposium on Computer-Communications Networks and Teletraffic, The Polytechnic Institute of Brooklyn, April, 1972.

23. Farber,D.J. and Larson,K.C., "The Structure of a Distributed Computing System -- Software", Symposium on Computer-Communications Networks and Teletraffic, The Polytechnic Institute of Brooklyn, April, 1972.

24. Flynn,M.J., "Very High Speed Computing Systems", Proc. IEEE, Vol.54, (December,1966), pp.1901-1909.

25. Flynn,M.J., "Shared Internal Resources in a Multiprocessor", Information Processing 71, Proc. IFIP Congress 1971, pp.565-569.

26. Flynn,M.J., "Some Computer Organizations and Their Effectiveness", IEEE Trans. Computers, Vol.C-21, No.9, (September,1972), pp.948-960.

27. Flynn,M.J. and Podvin,A., "Shared Resource Multiprocessing", Computer, Vol.5, No.2, (March/April,1972), pp.20-28.

28. Flynn,M.J., Podvin,A., and Shimizu,K., "A Multiple Instruction Stream Processor with Shared Resources", in Parallel Processor Systems, Technologies and Applications, L.C. Hobbs et al. (eds.), Spartan Books, 1970.

29. Foster,C.C., Computer Architecture, Van Nostrand Reinhold Co., 1970.

30. Gear,W.C., Computer Organization and Programming, McGraw-Hill 1969.

31. Giddings,G.M., "Cost Optimization in Multiprocessor Computer Systems", Ph.D. Thesis, U. of California, Berkeley, 1969.

32. Goldstine,H.H. and von Neumann,J., "On the Principles of Large Scale Computing Machines", unpublished, 1946, in Collected Works of John von Neumann, A.H.Taub (ed.), Vol.5, pp.1-32, Pergamon Press, 1961.

33. Goldstine,H.H. and von Neumann,J., "Planning and Coding Problems for an Electronic Computing Instrument", Part II, Vols.1-3, Rept. prepared for U.S.Army Ordnance Dept., 1947, in Collected Works of John von Neumann, A.H.Taub (ed.), Vol.5, pp.80-235, Pergamon Press, 1961.

34. Gonzalez,M.J. and Ramamoorthy,C.V., "Program Suitability for Parallel Processing", IEEE Trans. Computers, Vol.C-20, No.6, (June,1971), pp.647-654.

35. Gonzalez,M.J. and Ramamoorthy,C.V., "Parallel Task Execution in a Decentralized System", IEEE Trans. Computers, Vol.C-21, No.12, (December,1972), pp.1310-1322.

36. Graham,W.R., "The Impact of Future Developments in Computer Technology", Computers and Structures, Vol.1, Nos.1/2, (August,1971), pp.311-321.

37. Hellerman,H., Digital Computer System Principles, McGraw-Hill, 1967.

38. Hellerman,H. and Smith,H.J., "Throughput Analysis of Some Idealized Input, Output and Compute Overlap Configurations", Computing Surveys, Vol.2., No.2., (June,1970), pp.111-118.

39. Holland,J.H., "A Universal Computer Capable of Executing an Arbitrary Number of Subprograms Simultaneously", in Essays on Cellular Automata, (1970), pp.264-276. (Reprinted from Proc. EJCC 1959, pp.108-113.)

40. Holland,S.A. and Purcell,C.J., "The CDC STAR-100 -- A Large Scale Network Oriented Computer System", Proc. Sixth IEEE Internatl. Computer Society Conference, (September,1971), pp.55-56.

41. Ibbett,R.N., "The MU5 Instruction Pipeline", The Computer Journal, Vol.15,No.1,(February 1972), pp.42-50.

42. IBM Journal of Research and Development, Vol.11, No.1, (January,1967).

43. IBM System/360 OS: PL/I Language Specifications, Form C28-6571, (1965).

44. Juliussen,J.E. and F.J.Mowle, "Multiple Microprocessors with Common Main and Control Memories", IEEE Trans. on Computers, Vol.C-22,No.11,(November 1973),pp.999-1007.

45. Katzan,H., Computer Organization and the System 370, Van Nostrand Reinhold Co., New York, 1971.

46. Koczela,L.J., "Study of Spaceborne Multiprocessing", Final Report Phase II, Vol. II, C6-1476.22/23, North American Rockwell Corp., Autonetics Div., Anaheim, Cal., May,1968.

47. Koczela,L.J., "The Distributed Processor Organization", in Advances in Computers, F.L. Alt and M. Rubinoff (eds.), Vol.9, (1968), Academic Press, pp.285-353.

48. Kuck,D.J., "ILLIAC IV Software and Application Programming", IEEE Trans. Computers, Vol.C-17, No.8, (August,1968), pp.758-770.

49. Kuck,D.J., Muraoka,Y. and Chen,S.-C., "On the Number of Operations Simultaneously Executable in FORTRAN-like Programs and Their Resulting Speedup", IEEE Trans. Computers, Vol.C-21, No.12, (December,1972), pp.1293-1310.

50. Lehman,A., "A Survey of Problems and Preliminary Results Concerning Parallel Processing and Parallel Processors", Proc. IEEE, Vol.54, No.12, (December,1966), pp.1889-1901.

51. Lorin,H., Parallelism in Hardware and Software: Real and Apparent Concurrency, Prentice-Hall, 1972.

52. Mallach,E.G., "Job-Mix Modelling and System Analysis of an Aerospace Multiprocessor", IEEE Trans. Computers, Vol.C-21, No.5, (May,1972), pp.446-454.

53. Mitchell,J., et.al., "Multiprocessor Performance Analysis", AFIPS 1974 NCC, pp.399-403.

54. Morenoff,E., et al., "Four-way Parallel Processor Partition of an Atmospheric Primitive-Equation Prediction Model", Proc. AFIPS 1971 SJCC, Vol.38, pp.39-48.

55. Murtha,J.C., "Highly Parallel Information Processing Systems", in Advances in Computers, F.L.Alt and M.Rubinoff (eds.), Vol.7, Academic Press, New York, 1966, pp.1-116.

56. Nielsen,N.R., "Controlling a Real-Time Parallel Processing Computer", Simulation, Vol.17, No.3, (September,1971), pp.97-103.

57. Ramamoorthy,C.V. and Gonzalez,M.J., "A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs", Proc. AFIPS 1969 FJCC, Vol.35, pp.1-15.

58. Ramamoorthy,C.V. and Gonzalez,M.J., "Subexpression Ordering in the Execution of Arithmetic Expressions", Comm. ACM, Vol.14, No.7, (July,1971), pp.479-485.

59. Ravi,C.V., "On the Bandwidth and Interference in Interleaved Memory Systems", IEEE Trans. Computers, Vol.C-21, No.8, (August,1972), pp.899-901.

60. Regis,R.C., "Multiserver Queueing Models of Multiprocessing Systems", IEEE Trans. on Computers, Vol.C-22,No.8,(August 1973), pp.736-745.

61. Reigel,E.W., "Parallelism Exposure and Exploitation", in Parallel Processor Systems, Technologies and Applications, L.C.Hobbs et al. (eds.), Spartan Books, 1970.

62. Riorden,J., An Introduction to Combinatorial Analysis, Wiley and Sons,1958.

63. Roberts,L.G. and Wessler,B.D., "Computer Network Development to Achieve Resource Sharing", Proc. AFIPS 1970 SJCC, Vol.36, pp.543-549.

64. Rosen,S., "Electronic Computers: A Historical Survey", Computing Surveys, Vol.1, No.1, (March,1969), pp.7-36.

65. Skinner,C.E. and J.R.Asher, "Effect of Storage Contention on System Performance", IBM Systems J., Vol.8,No.4, (1969), pp.319-333.

66. Strecker,W.D., "An Analysis of the Instruction Execution Rate in Certain Computer Structures", Ph.D. Thesis, Carnegie-Mellon University, 1970.

67. Thornton,J.E., Design of a Computer: The Control Data 6600 Scott, Foresman and Co., Illinois, 1970.

68. Thornton,J.E., "Parallel Operation in the Control Data 6600", Proc. AFIPS 1964 FJCC, Vol.26, Pt.2, pp.33-40.

69. Ware,W.H., "The Ultimate Computer", IEEE Spectrum, Vol.1, No.3, (March,1972), pp.84-91.

70. Watson,W.J. and H.M.Carr, "Operational Experiences with the TI Advanced Scientific Computer", Proc. AFIPS 1974 NCC, pp.389-397.

71. Winder,R.O., "A Data Base for Computer Performance Evaluation", Computer, Vol.No.3,(March 1973),pp.25-29.